

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E
MATEMATICA APPLICATA



Project work - Real time systems and applications

Rover

Membri del gruppo

Nome e cognome	Matricola	E-mail
Adinolfi Teodoro	0622701902	t.adinolfi2@studenti.unisa.it
Amato Emilio	0622701903	e.amato16@studenti.unisa.it
Bove Antonio	0622701898	a.bove57@studenti.unisa.it
Guarini Alessio	0622702042	a.guarini7@studenti.unisa.it

ANNO ACCADEMICO 2023/2024

Indice

1 Architettura e schema di collegamento	3
1.1 Architettura di alto livello del sistema Rover	3
1.1.1 Sensori e attuatori della Board_1	3
1.1.2 Sensori della Board_2	4
1.1.3 Componenti condivisi	4
2 Dalle specifiche ai requisiti	5
3 Activity Diagrams	9
3.1 Funzionamento standard	9
3.1.1 Inizializzazione del sistema Rover	10
3.1.2 Avanzamento	11
3.1.3 Retromarcia standard	13
3.1.4 Retromarcia alternativa	15
3.1.5 Rotazione	17
3.1.6 Sterzata	19
3.1.7 Avanzamento da fermo in presenza di ostacoli	21
3.1.8 Avanzamento con comparsa di ostacoli	23
3.2 Funzionamento in modalità degradata	25
3.2.1 Spostamento a comando singolo	27
3.2.2 Spostamento a comandi combinati	29
3.3 Funzionamento in modalità di emergenza	31
3.3.1 Raggiungimento stato di emergenza	32
4 Definizione dello Stato	34
4.1 Definizione dello stato	34
5 Dimensionamento del sistema Rover	38
5.1 Calcolo dei Tempi per le Attività del Firmware	38
5.1.1 Lettura dei Sensori	38
5.1.2 Scambio delle Informazioni tra le Board	44
5.1.3 Attuazione	49
5.2 Progettazione dei task	50
5.2.1 Minimizzazione delle Dipendenze	50
5.2.2 Mapping del ciclo di controllo e sincronizzazione intra-board	51
5.2.3 Sincronizzazione inter-board	52
5.3 Scelta dell'algoritmo di scheduling	53

INDICE

5.3.1	Dimostrazione di schedulabilità	53
5.4	Spazio minimo di frenata	56
5.4.1	Un problema implementativo	57
5.4.2	Scheduling finale	58
5.4.3	Una soluzione alternativa al nuovo protocollo presentato	59
6	Macchina a stati	60
6.1	Approccio Generale	60
6.2	BOARD_1	62
6.2.1	Inizializzazione	68
6.2.2	Esecuzione	68
6.3	BOARD_2	76
6.3.1	Inizializzazione	78
6.3.2	Esecuzione	78
6.4	Board_1 vs Board_2: Emergenza	86
6.5	Alcuni esempi di funzionamento	88
6.5.1	Scenario 1: avanzamento nella direzione <i>avanti-destra</i> con ostacolo improvviso a destra	91
6.5.2	Scenario 2: accensione dei led	95
6.5.3	Scenario 3: aumento della temperatura	99
7	Firmware	103
7.1	Common Drivers	103
7.2	Primary Board	104
7.2.1	Pinout	104
7.3	Secondary Board	105
7.3.1	Pinout	105
7.4	Single Board	105
7.4.1	Caratteristiche Principali	105
8	Appendice A	108
8.1	Prototipazione del supporto HCSR04	108
8.1.1	Modellazione	108
9	Pinout e Schema di collegamento	111

Capitolo 1

Architettura e schema di collegamento

1.1 Architettura di alto livello del sistema Rover

Come illustrato in Figura 1.1, l'architettura del sistema Rover è sviluppata su due dispositivi, Board_1 e Board_2, ovvero due STM32-NUCLEO-F401RE con sensori specifici assegnati a ciascuna board.

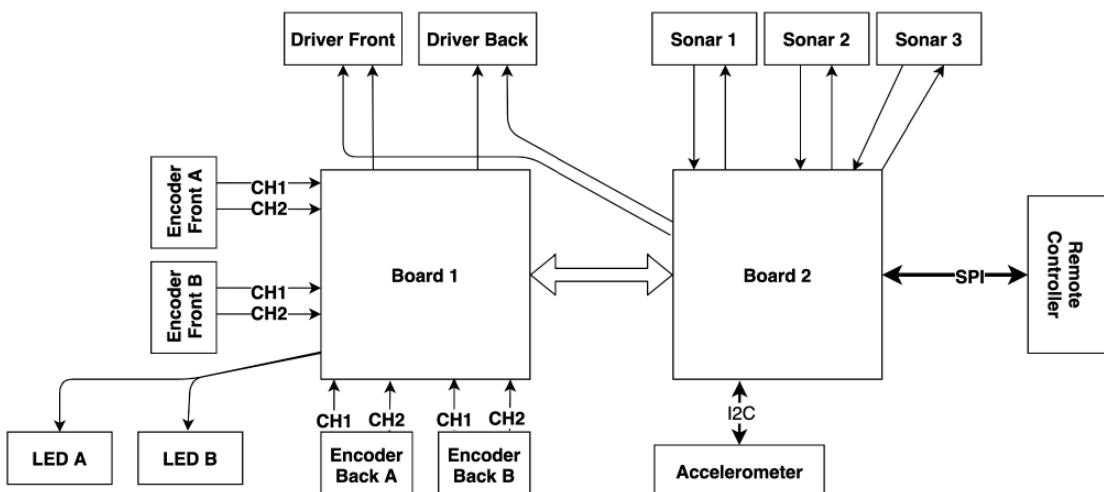


Figura 1.1: Architettura di alto livello del sistema

Importante notare è che alcuni sensori non sono interconnessi tra le due board, pertanto è fondamentale implementare un protocollo di comunicazione che permetta lo scambio di informazioni specifiche tra i dispositivi. Nel proseguo del documento, faremo riferimento alle board e ai relativi componenti con i nomi specificati nell'architettura.

1.1.1 Sensori e attuatori della Board_1

La Board_1 è equipaggiata con i seguenti sensori e attuatori:

- **Encoder:** ciascuno dei quattro motori brushed indipendenti 36GP-540-51 è dotato di un encoder a due canali con una risoluzione di 12 PPR e una frequenza di 800kHz
- **Led:** due moduli LED A4WD3 con luci rosse per la frenata e bianche ad alta potenza
- **Motor Driver:** due motor-driver Sabertooth 2X12 per il controllo dei quattro motori

1.1.2 Sensori della Board_2

La Board_2 integra una serie di sensori per la navigazione e il controllo del Rover:

- **Accelerometro e Giroscopio:** un MPU-6050 fornisce dati sull'accelerazione e l'orientamento
- **Sonar:** tre sensori sonar HCSR04 sono posizionati frontalmente e ai due lati per rilevare ostacoli, permettendo al Rover di rallentare o arrestarsi autonomamente
- **Remote Controller:** controllo remoto del Rover attraverso un DualShock 2, che comunica via Bluetooth con la board

1.1.3 Componenti condivisi

Alcuni componenti sono accessibili e gestiti da entrambe le board:

- **Motor Driver:** entrambe le board possono interagire con i motor driver Sabertooth 2X12 per la regolazione dei quattro motori

Capitolo 2

Dalle specifiche ai requisiti

In questo capitolo, ci concentreremo sulla trasformazione delle specifiche tecniche in requisiti funzionali. In particolare, lo strumento che utilizzeremo sono le *User Stories*, che aiutano a garantire che la progettazione del sistema rispetti le esigenze finali, fornendo anche un punto di riferimento utile per la fase di testing.

Si riportano di seguito le *User Stories* individuate.

User story 1

Come utente,
voglio poter inviare comandi tramite un controller PS2,
in modo da controllare il Rover a distanza.

User story 2

Come utente,
voglio utilizzare il controller PS2 in modalità analogica,
in modo da avere una risposta più fluida e precisa nella gestione dei
movimenti.

User story 3

Come utente,
voglio che l'inclinazione delle levette sul controller PS2 indichi la velocità e la
direzione del rover,
così da avere un controllo intuitivo del suo movimento.

User story 4

Come utente,
voglio usare i pulsanti del controller per azioni specifiche come lo stop di
emergenza e l'accensione di LED,
per avere un controllo diretto su tali funzioni.

User story 5

Come utente,
voglio che il Rover si muova in linea retta ad una velocità predefinita,
per poterlo dirigere facilmente verso una destinazione specifica.

User story 6

Come utente,
voglio che il Rover possa ruotare attorno al suo baricentro,
per effettuare manovre in spazi ristretti.

User story 7

Come utente,
voglio che il Rover legga i dati dai sensori,
per monitorare l'ambiente circostante e assicurare il funzionamento ottimale.

User story 8

Come utente,
desidero che il Rover usi i sensori ultrasound per identificare ostacoli almeno a
3 metri di distanza,
per prevenire collisioni durante il movimento.

User story 9

Come utente,
voglio che il Rover entri in uno stato sicuro "Motori Fermi" se identifica un
ostacolo,
per evitare incidenti.

User story 10

Come utente,
desidero una frenata dolce ('smooth') in presenza di ostacoli almeno a 3 metri
di distanza,
per arrestare il Rover in modo più controllato.

User story 11

Come utente,
voglio che il Rover esegua una frenata di emergenza e ruoti verso un lato
libero se identifica un ostacolo improvviso,
per evitare collisioni immediate.

User story 12

Come utente,
desidero che il Rover non avanzi se identifica un ostacolo mentre è fermo,

Dalle specifiche ai requisiti

per garantire sicurezza prima di iniziare il movimento.

User story 13

Come utente,
voglio che il Rover possa ruotare di 180° e avanzare in marcia indietro se non ci sono ostacoli (retromarcia alternativa),
per effettuare manovre in spazi limitati.

User story 14

Come utente,
desidero poter sbloccare la retromarcia alternativa tramite una combinazione di pulsanti,
per attivarla solo in caso di necessità

User story 15

Come utente,
voglio che il Rover raggiunga uno stato sicuro e si fermi in caso di gravi malfunzionamenti,
per prevenire danni ulteriori

User story 16

Come utente,
voglio che il Rover possa continuare a muoversi in modalità degradata in caso di malfunzionamenti parziali,
per mantenere una certa operatività anche in condizioni non ottimali

User story 17

Come utente,
voglio che il Rover imposti un limite sulla velocità massima quando è in modalità degradata, per ridurre il rischio di danni ulteriori al sistema o all'ambiente circostante

User story 18

Come utente,
voglio che il Rover si muova solo avanti e indietro ed effettui le rotazioni intorno al baricentro quando è in modalità degradata,
per semplificare la sua manovrabilità in condizioni non ottimali

Dalle user stories precedentemente individuate per il controllo e la gestione del Rover, possiamo derivare una serie di vincoli real-time, cruciali per garantire le funzionalità e la sicurezza del sistema. In particolare, possiamo elencare i seguenti vincoli:

- *Vincolo 1*

- **Vincolo:** risposta immediata ai cambiamenti di inclinazione delle levette
 - **Descrizione:** il sistema deve essere in grado di rilevare e reagire rapidamente ai cambiamenti nell'inclinazione delle levette, traducendo questi input in variazioni di velocità e direzione in tempo reale
- *Vincolo 2*
 - **Vincolo:** risposta immediata alla pressione dei pulsanti
 - **Descrizione:** l'azione associata alla pressione di un pulsante (es. stop di emergenza) deve essere eseguita immediatamente, senza ritardi percepibili
 - *Vincolo 3*
 - **Vincolo:** sincronizzazione dei motori
 - **Descrizione:** i quattro motori del Rover devono essere sincronizzati, per evitare deviazioni dal percorso previsto
 - *Vincolo 4*
 - **Vincolo:** rilevamento in tempo reale degli ostacoli
 - **Descrizione:** i sensori devono identificare gli ostacoli in tempo reale e il sistema deve elaborare questi dati il prima possibile per prendere decisioni rapide sullo stato fisico del Rover
 - *Vincolo 5*
 - **Vincolo:** monitoraggio continuo e identificazione rapida dei malfunzionamenti
 - **Descrizione:** il sistema deve costantemente monitorare lo stato dei suoi componenti e passare in uno stato sicuro immediatamente quando si rileva un malfunzionamento

Capitolo 3

Activity Diagrams

Il presente capitolo è dedicato all'esposizione e all'analisi degli scenari operativi del sistema Rover, delineando le diverse modalità di comportamento in risposta a specifiche condizioni operative. Attraverso una serie di activity diagrams, si illustrerà la sequenza di eventi e azioni che caratterizzano il funzionamento standard del sistema, nonchè le modalità operative alternative quali quelle di funzionamento degradato e di emergenza che si attivano rispettivamente in presenza di specifiche situazioni che saranno più avanti dettagliate.

3.1 Funzionamento standard

In questa sezione, saranno dettagliate le procedure che rappresentano il funzionamento ottimale del sistema Rover. Le precondizioni assunte includono l'integrità di tutti i componenti hardware, che sono rispettivamente

- i tre sensori ad ultrasuoni *HCSR04*
- l'accelerometro *MPU-6050*
- il remote controller *PS2*
- i quattro motori *36GP-540-51*
- i due motor driver *Sabertooth 2x12*
- i due led *A4WD3 Red Brake & White High Power LED Module*
- le due MCU *STM32F401RE*

oltre che ad una corretta alimentazione elettrica, l'integrità dei componenti software e la piena funzionalità del sistema di comunicazione sottostante, teso a garantire coerenza tra i dati sensoriali e l'attuazione dei comandi in risposta alle interazioni fornite dall'utente.

3.1.1 Inizializzazione del sistema Rover

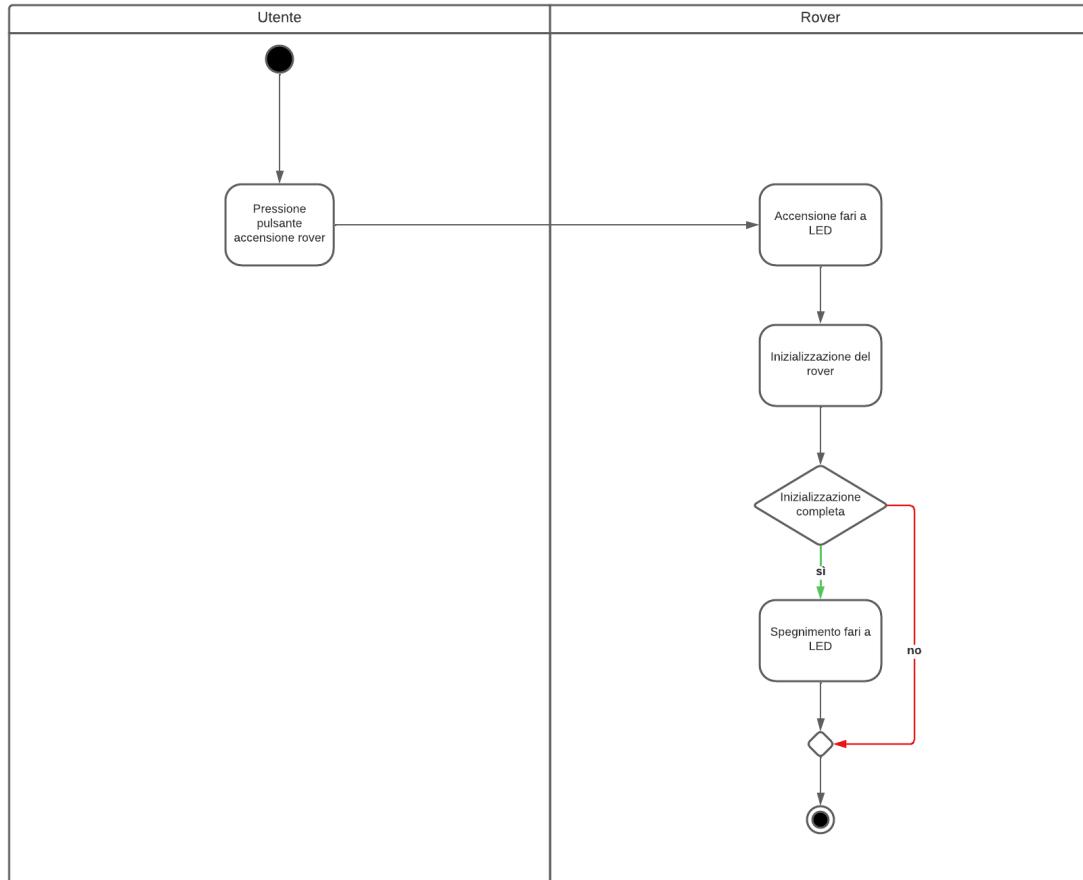


Figura 3.1: Scenario di inizializzazione del sistema Rover

Il seguente scenario descrive **l'inizializzazione del sistema Rover**, innescata dall'azione dell'utente che preme il pulsante di accensione. In risposta, i fari a LED si illuminano, segnalando l'avvio del processo. Il sistema procede con una serie di controlli diagnostici delle componenti hardware e software per assicurarsi che queste ultime siano operative.

Al termine il sistema verifica l'esito dell'inizializzazione e se positivo i fari si spengono, indicando la prontezza del sistema Rover per l'uso mentre se negativo restano accesi, segnalando la necessità di eventuali interventi. Di conseguenza il Rover non accetterà comandi fino al completamento e verifica con successo di un'ulteriore sequenza di avvio. Tutto questo processo è teso a garantire sicurezza e affidabilità prima dell'operatività del sistema.

3.1.2 Avanzamento

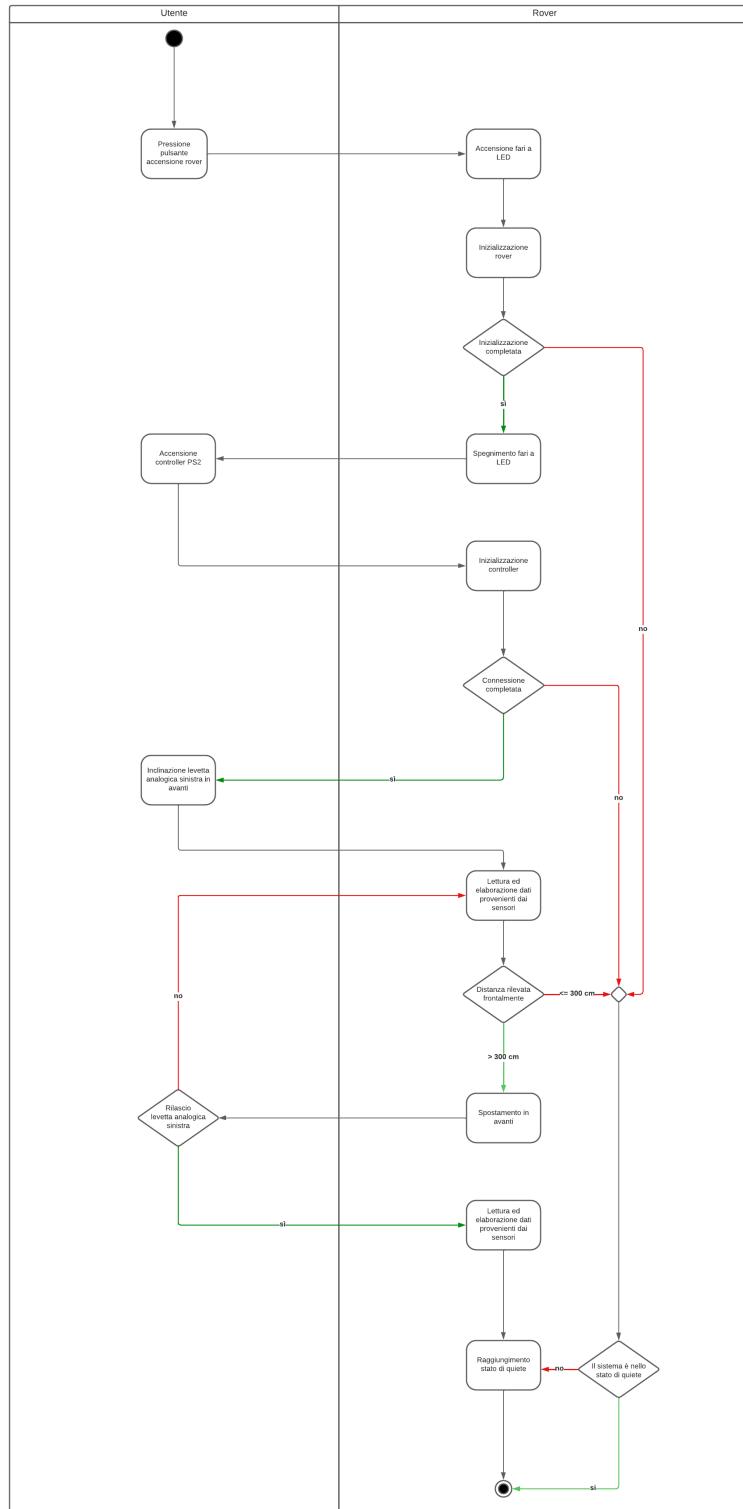


Figura 3.2: Scenario di avanzamento del sistema Rover

Activity Diagrams

L'activity diagram in esame articola il processo attraverso il quale un utente esterno ingaggia il **sistema rover per un movimento diretto in avanti**.

Inizialmente, l'utente attiva il rover mediante il pulsante di accensione, innescando la sequenza di inizializzazione precedentemente descritta. Al termine di questa, indicata dallo spegnimento dei fari a LED, il rover è configurato per accettare comandi esterni.

Successivamente, l'utente procede all'attivazione del controller PS2. Una volta stabilita la connessione, è possibile impartire comandi di direzione al rover inclinando in avanti la levetta analogica sinistra. Il rilascio della levetta comporta l'arresto immediato del rover. Durante il suo tragitto, il rover esegue letture e elaborazioni continue dei dati sensoriali per assicurare un'avanzamento sicuro. Se viene rilevata una distanza superiore ai 3 metri dagli ostacoli, il rover continua il movimento; altrimenti, si arresta per evitare potenziali collisioni.

Se il sistema si arresta, esso entra in uno stato di quiete in attesa di ulteriori comandi da parte dell'utente. Questo stato di quiete è mantenuto finché non si verificano nuove azioni da parte dell'utente o finché non si presentano altre condizioni che richiedono l'intervento del sistema. Questo scenario evidenzia l'interattività e reattività del sistema, mostrando come esso risponda nell'immediato all'input dell'utente garantendo allo stesso tempo la sicurezza attraverso il monitoraggio rispetto all'ambiente circostante.

3.1.3 Retromarcia standard

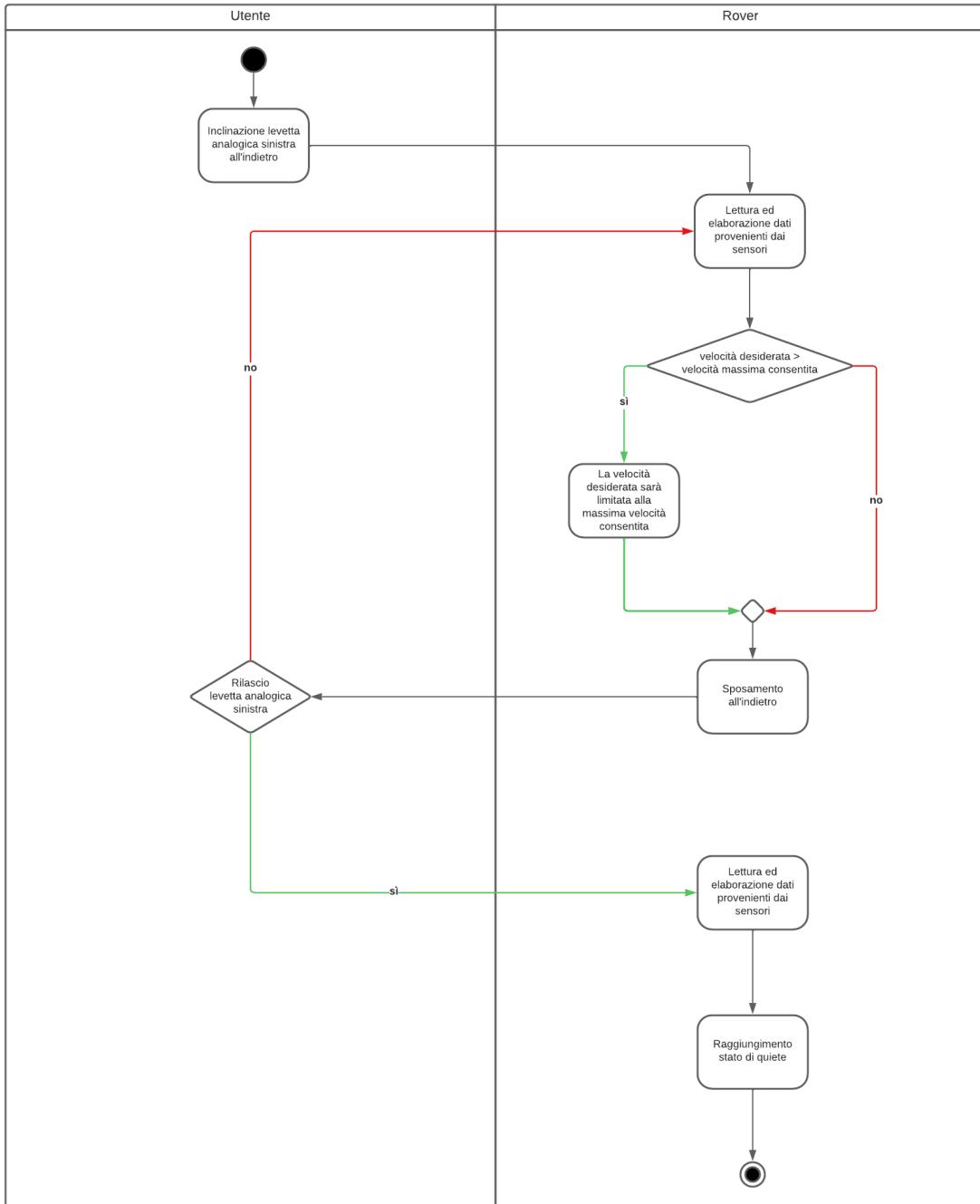


Figura 3.3: Scenario di arretramento standard del sistema Rover

Activity Diagrams

L'activity diagram presentato illustra il funzionamento del sistema Rover durante una **manovra di retromarcia in modalità standard**.

Dopo che l'utente inclina la levetta analogica sinistra all'indietro, il sistema Rover raccoglie ed elabora i dati provenienti dai sensori con la differenza che, rispetto alla manovra di avanzamento, durante la retromarcia non viene eseguita una verifica della distanza da eventuali ostacoli data l'assenza di sensori ad ultrasuoni orientati posteriormente sul Rover. Pertanto, il sistema si affida esclusivamente al giudizio dell'utente per la sicurezza durante il movimento all'indietro.

Precauzione

Per questo motivo, rispetto allo spostamento in avanti la velocità in retromarcia viene ulteriormente limitata, per rendere la manovra meno pericolosa e pertanto se la velocità desiderata supera la velocità massima consentita, il sistema Rover automaticamente limita la velocità alla soglia massima per mantenere la sicurezza operativa.

Una volta che l'utente rilascia la levetta analogica, il rover interrompe il processo di retromarcia e, dopo una lettura ed elaborazione dai sensori, raggiunge lo stato di quiete in attesa di nuovi comandi o azioni da parte dell'utente.

Tale scenario è finalizzato ad evidenziare che, rispetto all'avanzamento, data l'assenza di possibilità nel controllare la presenza di ostacoli nel senso di marcia posteriore si presuppone una maggiore vigilanza da parte dell'utente oltre che ad una limitazione delle prestazioni in termini di velocità rispetto a quanto consentito in marcia normale.

3.1.4 Retromarcia alternativa

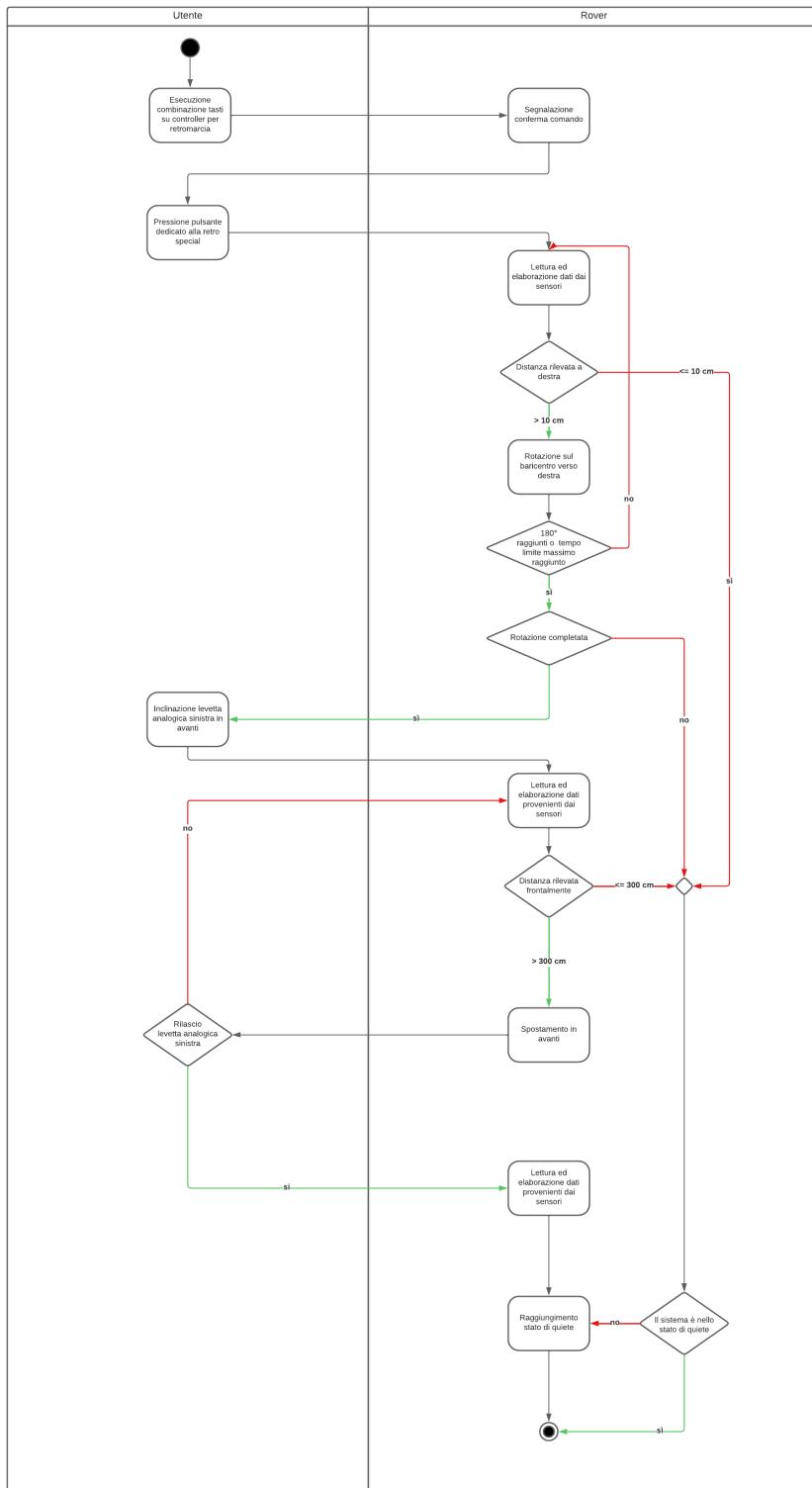


Figura 3.4: Scenario di arretramento alternativo del sistema Rover

L'activity diagram in esame descrive una **procedura di retromarcia alternativa** eseguibile per il sistema Rover. Tale modalità è distinta dalla **retromarcia standard** e può essere attivata esclusivamente quando il sistema si trova in uno stato di quiete, come accade all'interno dello scenario ivi descritto.

Per poter attivare tale modalità l'utente esegue la combinazioni di tasti sul controller ed, una volta confermato il comando, preme un pulsante dedicato a tale funzione. Successivamente, il sistema procede con la lettura e l'elaborazione dei dati provenienti dai sensori ed, una volta innescata la rotazione sul baricentro, se vengono rilevati ostacoli nel senso in cui è effettuata ovvero destra, entro 10 centimetri, il sistema interrompe la rotazione e la manovra di retromarcia, assicurando di evitare collisioni con oggetti che possono essere rilevati durante l'esecuzione del comando.

Nota

La scelta di fissare una soglia di 10 centimetri come limite minimo di sicurezza per la distanza dagli ostacoli durante la rotazione sul baricentro è stata presa seguendo un approccio basato sulle dimensioni fisiche del sistema e sulla dinamica della manovra. Attraverso valutazioni geometriche relative alla struttura del sistema ed ai suoi movimenti durante la rotazione sul baricentro, è stato determinato che un ostacolo deve essere posizionato a una distanza superiore a 7 centimetri per evitare il contatto e dunque la soglia di sicurezza di 10 centimetri utilizzata dà quel margine aggiuntivo che tiene conto delle incertezze.

Una volta innescata, se la rotazione di 180° viene completata o viene raggiunto il tempo massimo prestabilito per l'operazione, la manovra viene interrotta. Difatti, qualora la manovra venga interrotta per la seconda motivazione descritta, questo significa che la rotazione non è completa a causa di ostacoli non visibili dai sensori e dunque la presenza di angoli morti, fermandosi di conseguenza per evitare danni. Una volta che la rotazione è stata completata senza incongruenti, il sistema verifica la presenza di ostacoli frontalmente. Se presenti ad una distanza superiore alla soglia prestabilita, il sistema Rover inizia a muoversi in avanti, che rappresenta la direzione posteriore rispetto al suo posizionamento originale.

Infine, l'utente può interrompere la manovra di spostamento in qualsiasi momento rilasciando la levetta analogica sinistra. Dopo il rilascio, il sistema effettua ulteriori letture ed elaborazioni dai sensori e, una volta raggiunto lo stato di quiete, resta in attesa di nuovi comandi.

3.1.5 Rotazione

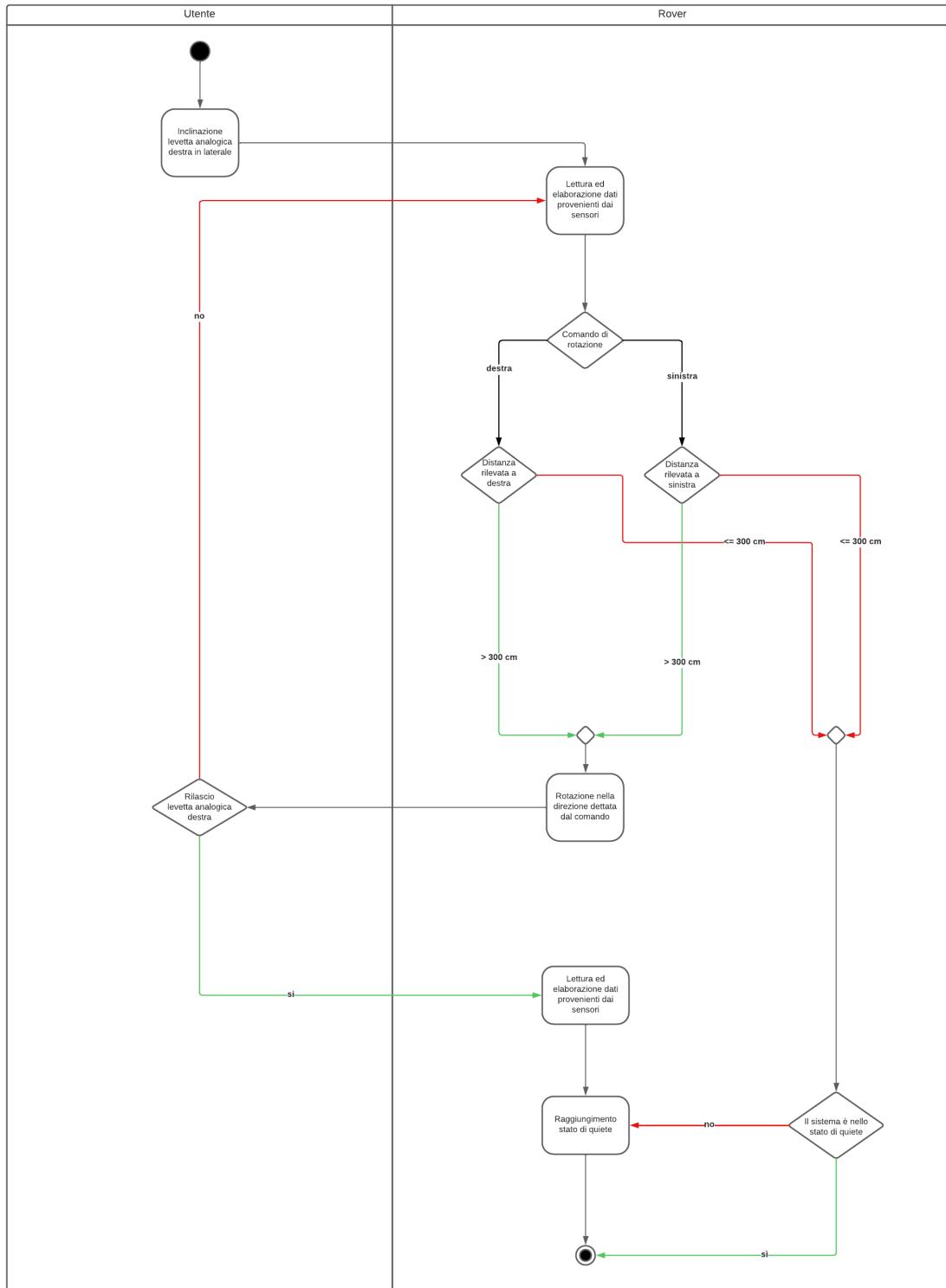


Figura 3.5: Scenario di rotazione del sistema Rover

Activity Diagrams

L'activity diagram delineato descrive l'operazione di **rotazione del sistema Rover attorno al proprio baricentro**. La descrizione di questa modalità operativa evidenzia la risposta del sistema ai comandi forniti dall'utente e le misure di sicurezza implementate per evitare collisioni durante la manovra.

Quando l'utente inclina la levetta analogica destra verso destra, il sistema controlla la fattibilità della rotazione in tale direzione, leggendo ed elaborando i dati dai sensori per determinare se ci sono ostacoli ed, in caso negativo, il sistema procede con la rotazione. In modo simile, se l'utente desidera ruotare il Rover verso sinistra, il sistema effettuerà le stesse valutazioni ma nella direzione opposta. La decisione di procedere con la rotazione è presa dunque solo se lo spazio è libero dagli ostacoli fino ad una certa distanza anche in questa direzione.

È importante notare che la sicurezza della manovra è assicurata dal controllo della distanza solo verso la direzione di rotazione indicata dall'utente. Nessun controllo è effettuato in direzione opposta, poiché si presuppone che la parte opposta alla rotazione del sistema resti entro i confini di un percorso precedentemente verificato come libero da ostacoli.

Una volta che la rotazione è completata o interrotta, per l'interruzione da parte dell'utente o per un ostacolo incontrato, il sistema si assicura di raggiungere uno stato di quiete prima di procedere con ulteriori comandi.

3.1.6 Sterzata

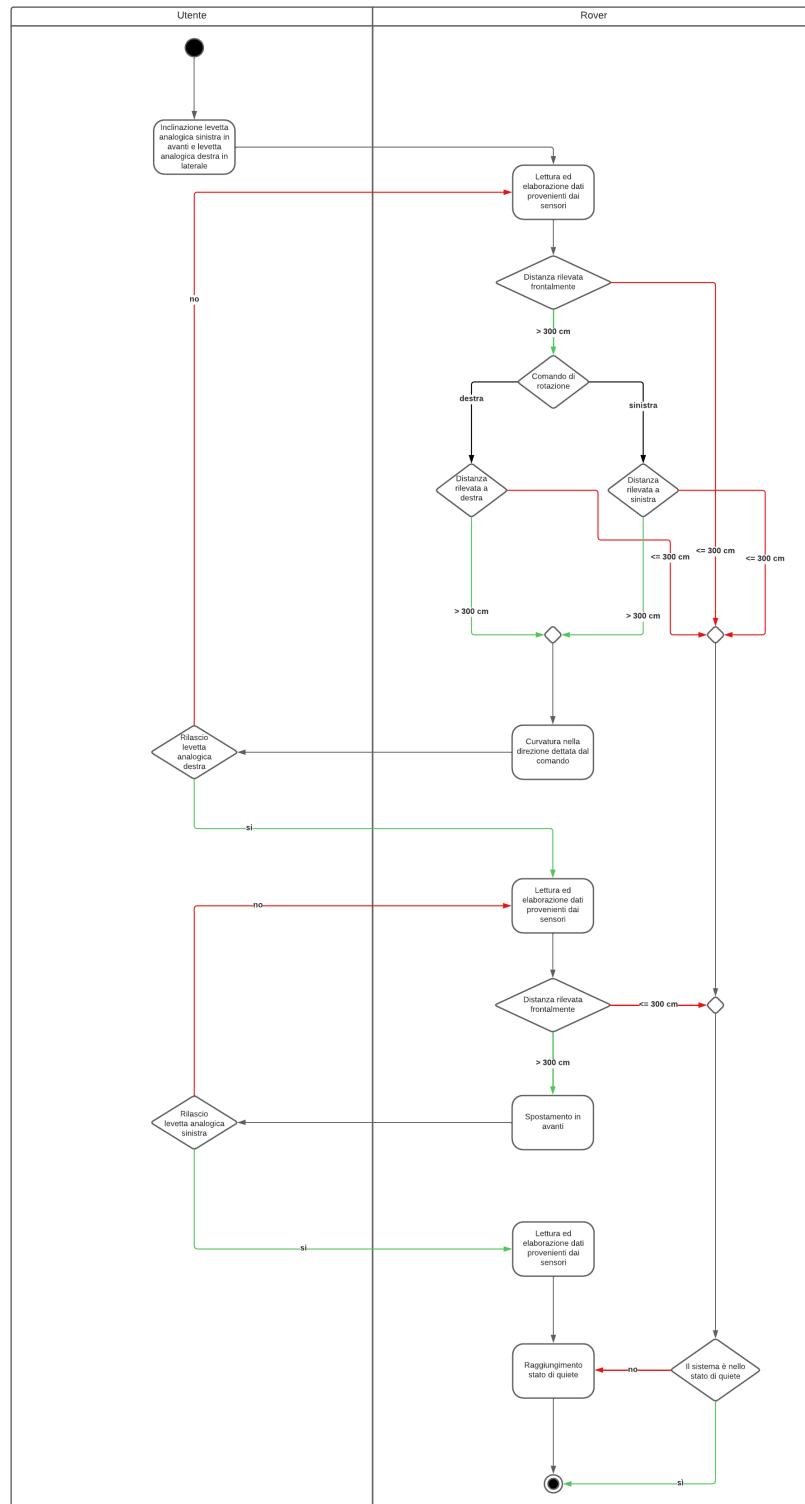


Figura 3.6: Scenario di sterzata del sistema Rover

Activity Diagrams

L'activity diagram in esame descrive il **comportamento del sistema Rover quando esegue una manovra di sterzata**, procedendo in avanti e ruotando contemporaneamente in una direzione determinata dall'utente.

In questo scenario, la manovra inizia con l'utente che inclina la levetta analogica sinistra in avanti e quella destra per sterzare. Il sistema Rover, prima di eseguire la manovra, effettua una verifica della distanza dagli ostacoli posizionati frontalmente. Il controllo dagli ostacoli con questa modalità operativa è più complesso rispetto alla semplice rotazione attorno al baricentro, poiché durante una sterzata il Rover deve considerare anche gli ostacoli presenti lateralmente nel nuovo percorso che si andrà a creare a seguito della variazione di direzione.

Se non sono presenti ostacoli nelle direzioni di interesse, il sistema esegue la curvatura nella direzione indicata, continuando a monitorare l'ambiente per assicurarsi che la distanza dagli ostacoli rimanga superiore alla soglia di sicurezza, interrompendo la manovra in caso contrario.

Il rilascio della levetta analogica da parte dell'utente interrompe la manovra e porta il sistema a uno stato di quiete, in attesa di ulteriori comandi.

3.1.7 Avanzamento da fermo in presenza di ostacoli

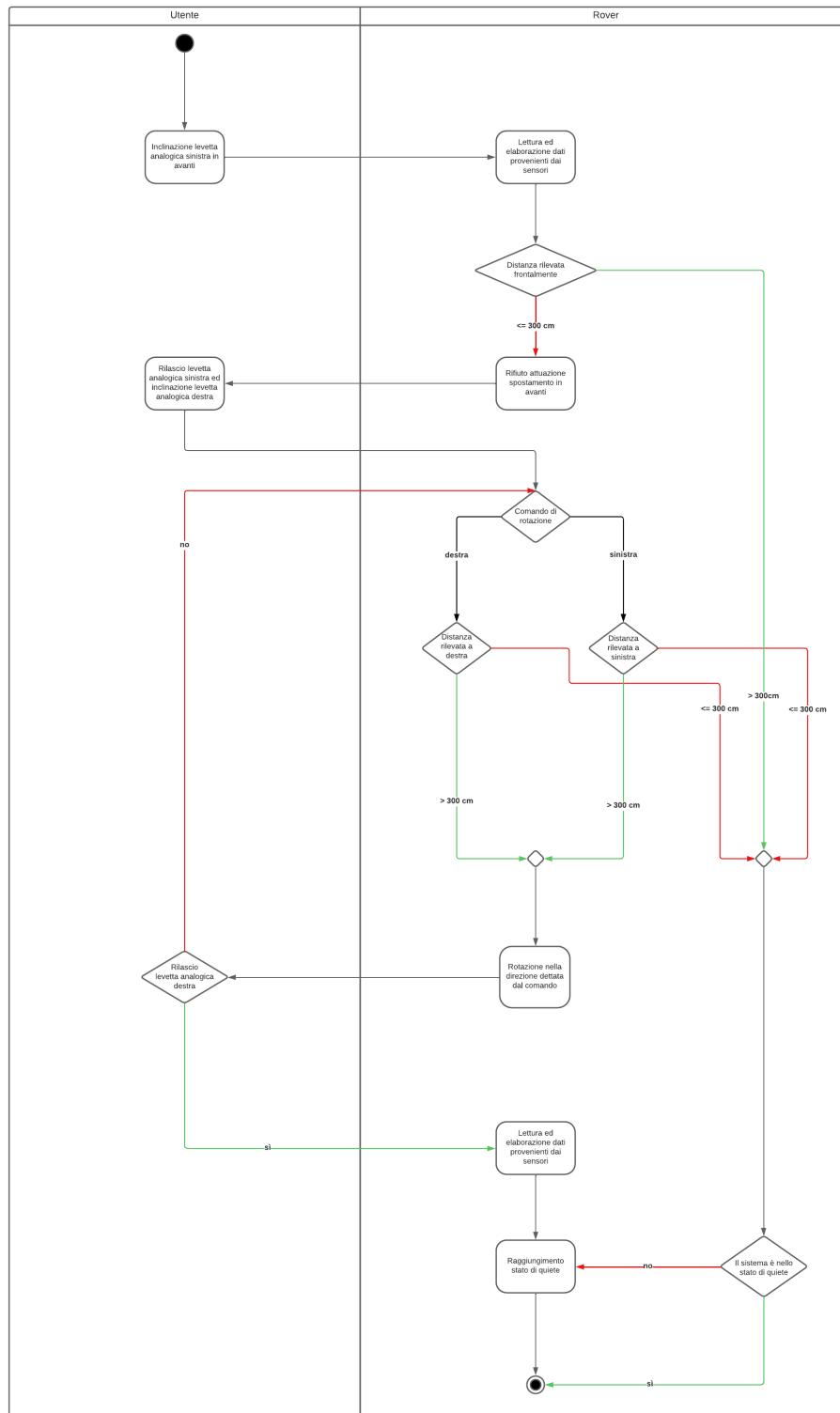


Figura 3.7: Scenario di avanzamento da fermo con ostacoli del sistema Rover

Activity Diagrams

L'activity diagram seguente rappresenta il **comportamento del sistema Rover quando si trova in uno stato di quiete davanti a uno ostacolo e l'utente tenta di farlo avanzare.**

In tale scenario, l'utente tenta di spostare il Rover in avanti inclinando la levetta analogica sinistra in tale direzione ma, a seguito di una lettura ed elaborazione dei dati provenienti dai sensori, nel rilevare la presenza di un ostacolo ad una distanza inferiore alla soglia critica prestabilita il Rover non esegue il comando.

In tal caso, come modellato dallo scenario in esame, l'utente nell'avere disponibili una o entrambe le direzioni laterali libere, procede prima con la rotazione e poi con lo spostamento eventuale in avanti al fine di aggirare l'ostacolo che impedisce l'avanzamento precedentemente richiesto.

Tale comportamento riflette la progettazione e l'aderenza alle specifiche progettuali del sistema, sottolineando la necessità di un ambiente operativo libero da ostacoli per la navigazione sicura del Rover.

3.1.8 Avanzamento con comparsa di ostacoli

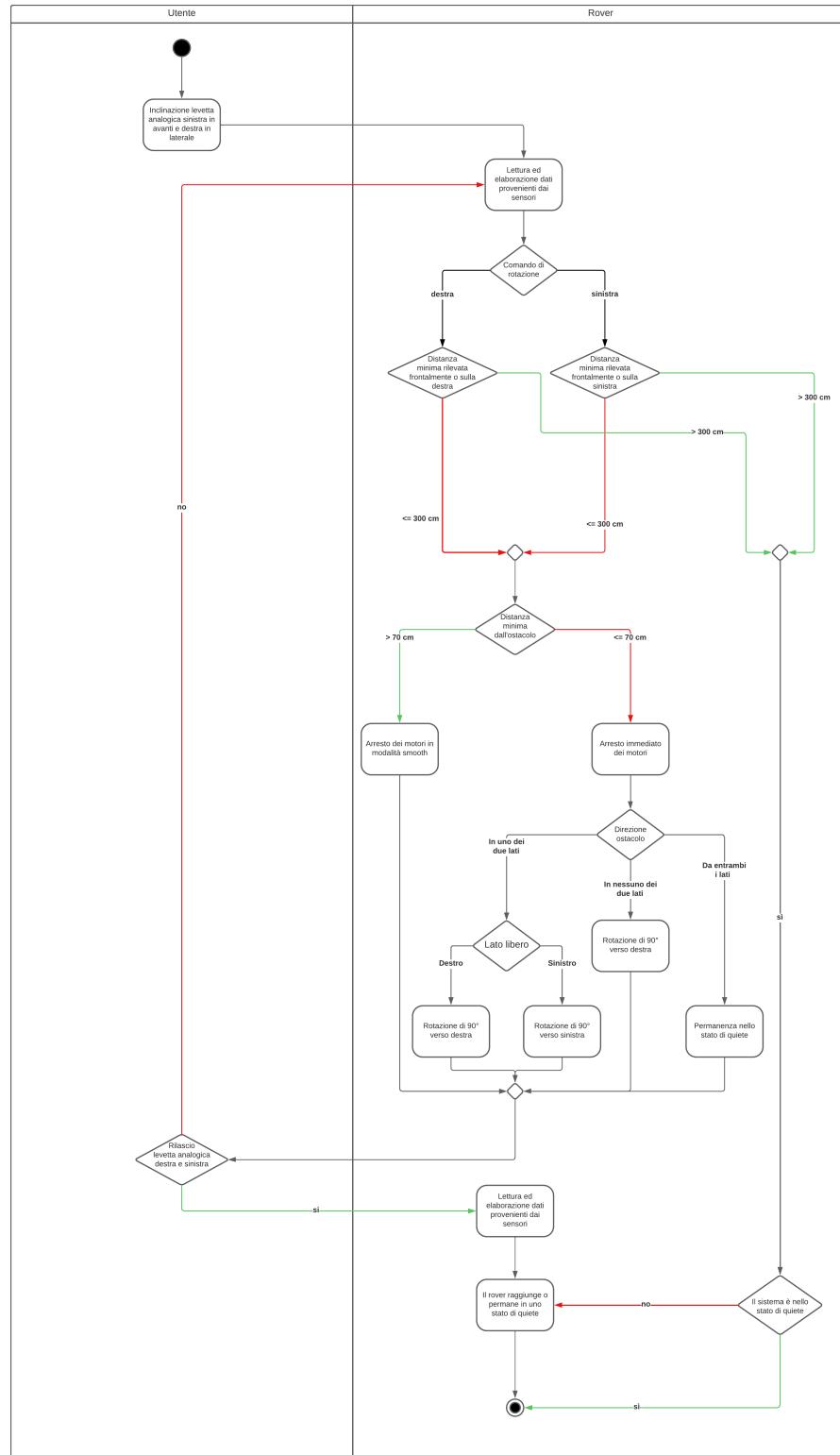


Figura 3.8: Scenario di avanzamento con comparsa di ostacoli del sistema Rover

Il seguente activity diagram illustra la logica di **gestione degli ostacoli con comparsa improvvisa durante l'avanzamento** del sistema Rover, il cui comportamento è modellato per reagire dinamicamente.

Lo scenario ha inizio con l'utente che inclina la levetta analogica sinistra in avanti e la levetta analogica destra in laterale per sterzare. Successivamente, il sistema effettua una lettura ed elaborazione dei dati dai sensori per monitorare la presenza di ostacoli lungo il percorso. Nel momento in cui la distanza minima rilevata nella direzione in cui è richiesto lo spostamento risulta inferiore alla soglia predefinita dei 300 cm, il Rover ne valuta il valore esatto per determinare il tipo di frenata necessaria:

1. **Frenata Immediata**: se la distanza dall'ostacolo è inferiore ai 70 cm, il Rover esegue un arresto immediato dei motori, azione necessaria al fine di evitare impatti con l'ostacolo
2. **Frenata Smooth**: se la distanza è inferiore ai 300 cm ma superiore ai 70 cm, il sistema esegue la frenata con una dinamica *smooth*, attraverso una decelerazione progressiva che consente al Rover di ridurre la velocità e fermarsi con cautela.

Dopo che il rover ha eseguito la frenata, se **smooth** allora si arresterà e permarrà in tale stato, altrimenti si comporterà nel seguente modo:

- **Con un lato libero**: se presente uno dei due lati liberi, il Rover si prepara a ruotare di 90 gradi verso quest'ultimo, permettendo così di navigare intorno all'ostacolo
- **Con entrambi i lati liberi**: se entrambi i lati sono liberi, il Rover per ruoterà di 90 gradi verso destra, scelta definita da progetto
- **Con nessun lato libero**: se il Rover è circondato da ostacoli e non ci sono lati liberi, rimane in uno stato di quiete.

Dopo aver eseguito la rotazione o, se necessario, essersi fermato completamente, il Rover procede con un'ulteriore serie di letture ed elaborazioni dei dati dai sensori, essenziali per confermare che il sistema si trovi in uno stato sicuro per riprendere le operazioni o per rimanere in attesa di ulteriori comandi.

3.2 Funzionamento in modalità degradata

All'interno di questa sezione si analizza il comportamento del sistema Rover quando questo non può operare nelle piene funzionalità a causa di anomalie nei componenti hardware o nei sensori. La modalità degradata è concepita per garantire la prosecuzione delle operazioni in sicurezza, con prestazioni ridotte fino a quando non sarà possibile effettuare un intervento di manutenzione. Il sistema Rover effettua un continuo monitoraggio dello stato delle sue componenti e, al manifestarsi di specifiche condizioni anomale, induce il passaggio alla modalità degradata. Di seguito sono dettagliate le circostanze che attivano tale modalità:

- **Failure della Board_1:** se l'MCU identificata nel sistema come *Board_1* presenta guasti o malfunzionamenti, il sistema entrerà in modalità degradata limitando la velocità massima del rover a causa dell'assenza di dati provenienti dagli encoder presenti sulla board interessata, gestendo l'esecuzione dei comandi di movimento del sistema in sicurezza
- **Malfunzionamento di un encoder:** in caso di guasto o mancata risposta di un encoder collegato ai motori gestiti da uno dei moduli *Sabertooth 2x12*, il sistema Rover continua la navigazione con una velocità limitata, consentendo un controllo più fine e riducendo la possibilità di un errore maggiore a causa dell'incertezza sul numero di giri a minuto esatti delle ruote
- **Malfunzionamento dell'accelerometro:** se l'accelerometro non fornisce dati o non funziona correttamente, il sistema rover entra in modalità degradata e la limitazione di velocità in questo scenario è finalizzata a evitare manovre aggressive e potenzialmente pericolose che richiedono una percezione accurata dell'orientamento e dell'accelerazione del sistema
- **Raggiungimento di una temperatura elevata:** quando il sistema di rilevamento della temperatura registra valori che superano una soglia prefissata di 35 gradi Celsius per un tempo consecutivo di 1 minuto, il sistema Rover rileva un rischio di surriscaldamento. La scelta di questa soglia specifica è motivata dalla temperatura massima operativa della batteria a ioni di litio utilizzata la quale può subire danni irreversibili o una diminuzione dell'efficienza quando esposta a temperatura superiore rispetto al limite operativo. Di conseguenza, si attua una diminuzione delle prestazioni in termini di velocità riducendo il rischio di generare ulteriore calore a causa dello sforzo dei motori e dell'elettronica di potenza.

Le misure attuate per preservare l'integrità del sistema sono essenziali per assicurare che, anche in presenza di problematiche tecniche come quelle precedentemente esposte, il sistema Rover possa raggiungere uno stato sicuro o completare il proprio task con un margine di sicurezza accettabile.

Activity Diagrams

Il sistema permette tramite il controller di effettuare spostamenti in avanti, all'indietro e verso entrambe le direzioni laterali, garantendo assenza di collisioni grazie ad un costante controllo nell'ambiente circostante rispetto agli ostacoli.

Activity Diagrams

3.2.1 Spostamento a comando singolo

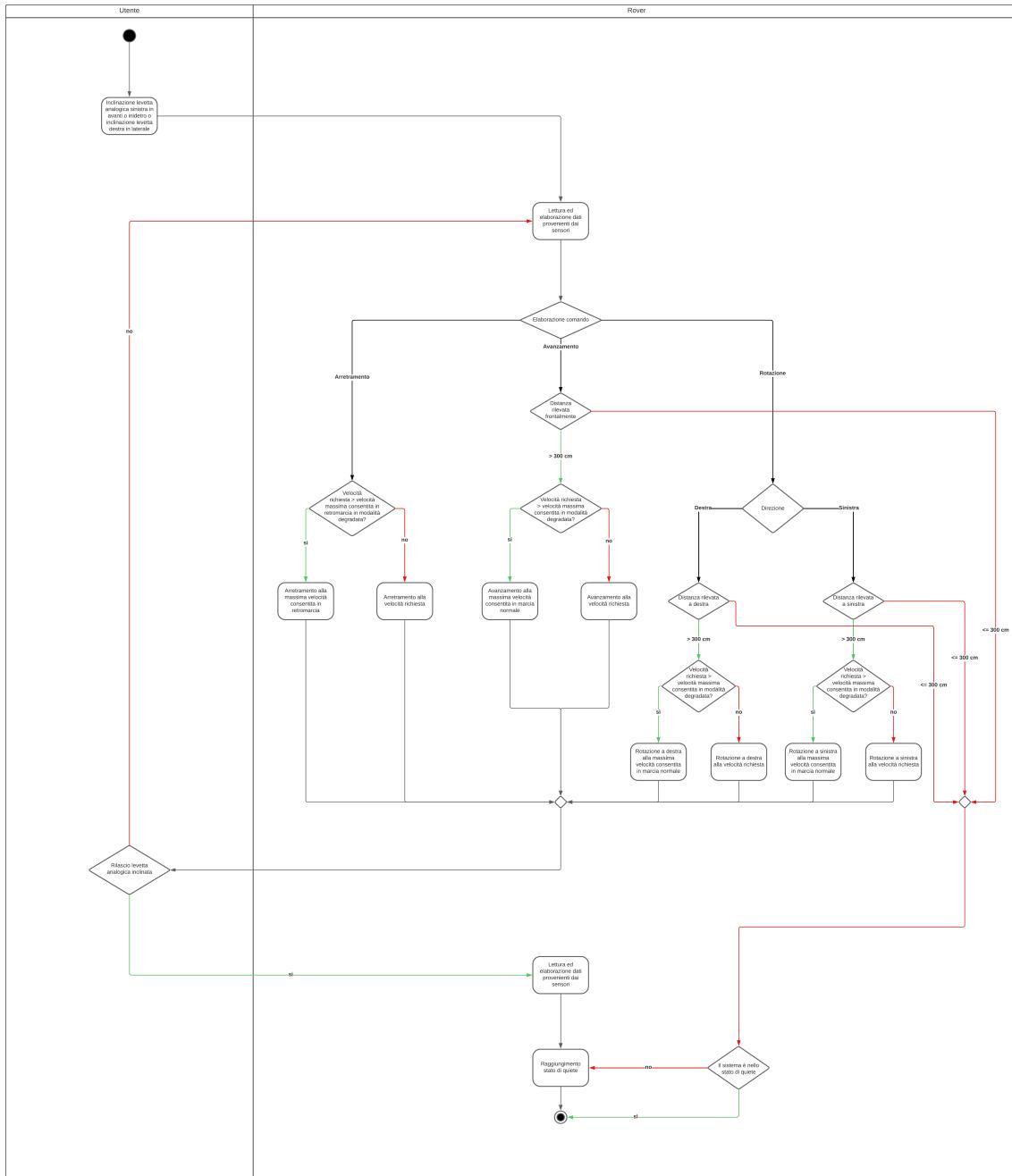


Figura 3.9: Scenario di spostamento a singolo comando del sistema Rover

Activity Diagrams

L'activity diagram presentato espone il comportamento del sistema Rover quando opera in **modalità degradata**, una condizione innescata da specifici fattori esposti nella sezione precedente i quali influenzano le prestazioni standard del sistema. In tale modalità, il Rover è soggetto a restrizioni operative in termini di velocità di movimento per garantire maggior sicurezza.

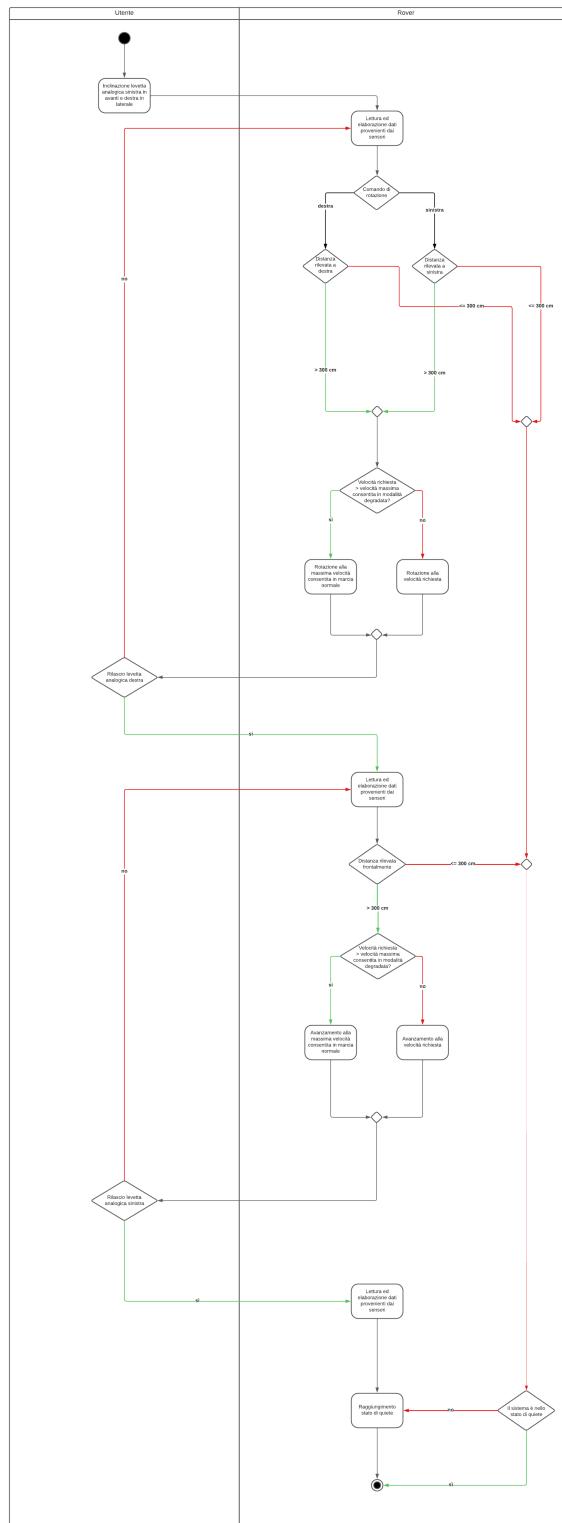
In caso di un comando di movimento anteriore da parte dell'utente, il sistema verifica prima che ci sia spazio sufficiente per eseguire la manovra ed in caso positivo procede con la verifica della velocità richiesta dall'utente, se eccede o meno la soglia massima consentita: sotto tale soglia il Rover procede con lo spostamento alla velocità desiderata; qualora la velocità sia superiore al limite imposto il sistema automaticamente limita al massimo consentito.

Similmente, per la rotazione, viene dapprima verificato che ci sia spazio a sufficienza per consentire la manovra e successivamente la velocità desiderata di esecuzione, con lo stesso effetto a seconda che sia al di sopra o meno della soglia massima consentita, per prevenire manovre brusche e potenzialmente pericolose.

Per il movimento in retromarcia, il sistema Rover attua una limitazione ancor più accentuata della velocità. Questa restrizione aggiuntiva riflette la mancanza di sensori posteriori per il rilevamento degli ostacoli, aumentando la necessità di una manovra sicura e controllata.

In tutti i casi, il Rover si porta in uno stato di quiete se non vi sono azioni ulteriori richieste dall'utente o se la distanza dagli ostacoli è tale per cui il sistema non possa spostarsi in sicurezza.

3.2.2 Spostamento a comandi combinati



Activity Diagrams

L'activity diagram proposto illustra il comportamento del sistema in **modalità degradata quando l'utente impedisce comandi multipli** che implicano un movimento combinato di avanzamento e rotazione. In questa situazione il sistema è progettato per assicurare operazioni sicure, limitando la complessità delle manovre e privilegiando l'azione singola che comporta minor controllo.

All'attuazione di comandi simultanei per avanzamento e rotazione, il sistema assegna priorità alla rotazione, scelta progettuale dettata dalla necessità di prevenire manovre non sicure a causa dall'assenza di una o più informazioni provenienti dai componenti guasti.

Quando l'utente muove entrambe le levette, il sistema inizia valutando la fattibilità in termini di presenza di ostacoli per la rotazione nella direzione desiderata, eseguendo una lettura ed elaborazione dei dati provenienti dai sensori: con una distanza superiore alla soglia dei 300 cm prestabilita la rotazione è permessa, altrimenti arresta il movimento per prevenire una collisione.

Dopo aver priorizzato e completato a rotazione, se la levetta analogica destra viene rilasciata mentre quella sinistra è ancora inclinata, il sistema procede con la valutazione del comando di avanzamento. Ancora una volta, il Rover verifica la distanza dagli ostacoli frontalmente e se lo spazio è sufficiente il movimento viene eseguito alla velocità al più massima consentita dalla modalità degradata mentre in caso contrario il Rover raggiunge lo stato di quiete, attendendo la rimozione dell'ostacolo o un nuovo comando da parte dell'utente.

Tale approccio mira a minimizzare i rischi e mantenere quanto più sicuro il sistema e l'ambiente operativo circostante, soprattutto in situazioni in cui le componenti del sistema sono limitate.

3.3 Funzionamento in modalità di emergenza

Questa sezione si focalizza sulla modalità di emergenza, un meccanismo di sicurezza che si attiva quando il sistema identifica condizioni che impediscono il funzionamento sicuro e controllato del sistema Rover. La modalità di emergenza si innesca automaticamente in risposta a specifici eventi con l'obiettivo di portare il sistema in uno stato sicuro, ovvero un arresto completo. Le condizioni che attivano questa modalità sono:

- **Failure della Board_2:** in caso di malfunzionamento dell'MCU principale denominata come *Board_2*, il sistema Rover interrompe immediatamente tutte le operazioni per prevenire decisioni errate e dunque azioni non coordinate
- **Failure di entrambe le board:** se entrambe le MCU presentano problemi, non avendo più controllo del sistema quest'ultimo si arresta per evitare comportamenti dannosi
- **Malfunzionamento degli encoder:** l'impossibilità di almeno due encoder nel fornire dati corretti porta il sistema Rover ad un arresto, prevenendo spostamenti basati su informazioni in termini di giri al minuto incomplete
- **Joypad Scarico:** se il controller remoto è scarico e quindi non in grado di inviare comandi, il sistema Rover cessa l'operatività per evitare azioni non intenzionali
- **Temperatura critica:** se il sistema Rover rileva una temperatura superiore ai 50 gradi celsius, si arresta e permane nello stato di emergenza finché non scende al valore operativo di riferimento. La soglia dei 50 gradi celsius è stata determinata in base alle caratteristiche operative della batteria a ioni di litio la quale prevede un range ottimale compreso tra 0 e 45 gradi celsius circa. Di conseguenza, la modalità di emergenza raggiunta superando questa soglia serve a prevenire eventuali danni dovuti al surriscaldamento e mantenere l'operatività entro i limiti di sicurezza prescritti per le celle della batteria.
- **Livello di carica della Batteria:** una carica eccessivamente bassa, riscontrata al di sotto del 23% del totale, comporta l'arresto del sistema Rover sia per preservare l'integrità della batteria che impedire spegnimenti improvvisi durante lo spostamento
- **Assenza di risposte dai Motor Controller:** l'assenza di feedback in relazione ai comandi forniti da uno o entrambi i motor driver innesca l'arresto del rover ed il raggiungimento dello stato di emergenza, al fine di evitare perdite di controllo sui motori stessi.

Dopo aver raggiunto lo stato di sicurezza a seguito di un'emergenza, il sistema Rover sospende l'elaborazione di qualsiasi comando in attesa che vengano ripristinate le condizioni operative normali. Nel caso in cui un tentativo di

riavvio non dovesse mitigare le condizioni di failure riscontrate, il sistema persiste nello stato di emergenza, ignorando eventuali ulteriori comandi. Questo meccanismo di sicurezza assicura che il Rover non riprenda le operazioni fino a quando non sia garantita la piena risoluzione dei problemi che hanno causato l'entrata in modalità di emergenza, risultando essenziale per preservare l'integrità del sistema e la sicurezza dell'ambiente operativo.

3.3.1 Raggiungimento stato di emergenza

L'activity diagram seguente descrive il comportamento del sistema rover in condizioni normali di funzionamento e il **passaggio automatico allo stato di emergenza** qualora si rilevi un failure tra quelle riportati nella precedente sezione.

In condizioni operative senza guasti, il Rover risponde all'utente attraverso i comandi imparatiti e, mentre esegue le azioni richieste come l'avanzamento sotto descritto, il sistema monitora continuamente il proprio stato assicurandosi di poter servire con sicurezza e correttezza tutti i comandi ricevuti. Qualora tale obiettivo non fosse correttamente raggiungibile, il sistema entra immediatamente in uno stato di emergenza, rilevando un failure.

Nota

Nello **stato di emergenza**, il Rover interrompe ogni operazione arrestando immediatamente i motori. Inoltre in questo stato il sistema rifiuta qualsiasi comando fornito dall'utente poiché la priorità è mantenere una condizione di sicurezza.

Raggiunto lo stato di emergenza, l'utente ha la possibilità di tentare un ripristino attraverso un pulsante dedicato. A seguito di questo tentativo di ripristino, si possono verificare due esiti distinti:

1. **Ripristino dal failure:** se il problema che ha causato il failure viene risolto, il sistema esce dallo stato di emergenza e torna operativo, divenendo nuovamente in grado di ricevere e eseguire comandi dall'utente, come mostrato nel diagramma con l'esempio di un comando di avanzamento
2. **Persistenza nello stato di emergenza:** se il tentativo di ripristino non risolve la causa del failure, il sistema Rover permane nello stato di emergenza, continuando a non accettare comandi e rimanendo fermo fino a quando non verrà risolto il problema.

L'activity diagram dimostra come il sistema garantisce operatività sicura anche in presenza di failure, attraverso il passaggio automatico allo stato di emergenza e come, attraverso un meccanismo di ripristino, il sistema possa tentare di tornare in uno stato funzionale, dando sempre priorità massima alla sicurezza, come evidenziato in ogni scenario.

Activity Diagrams

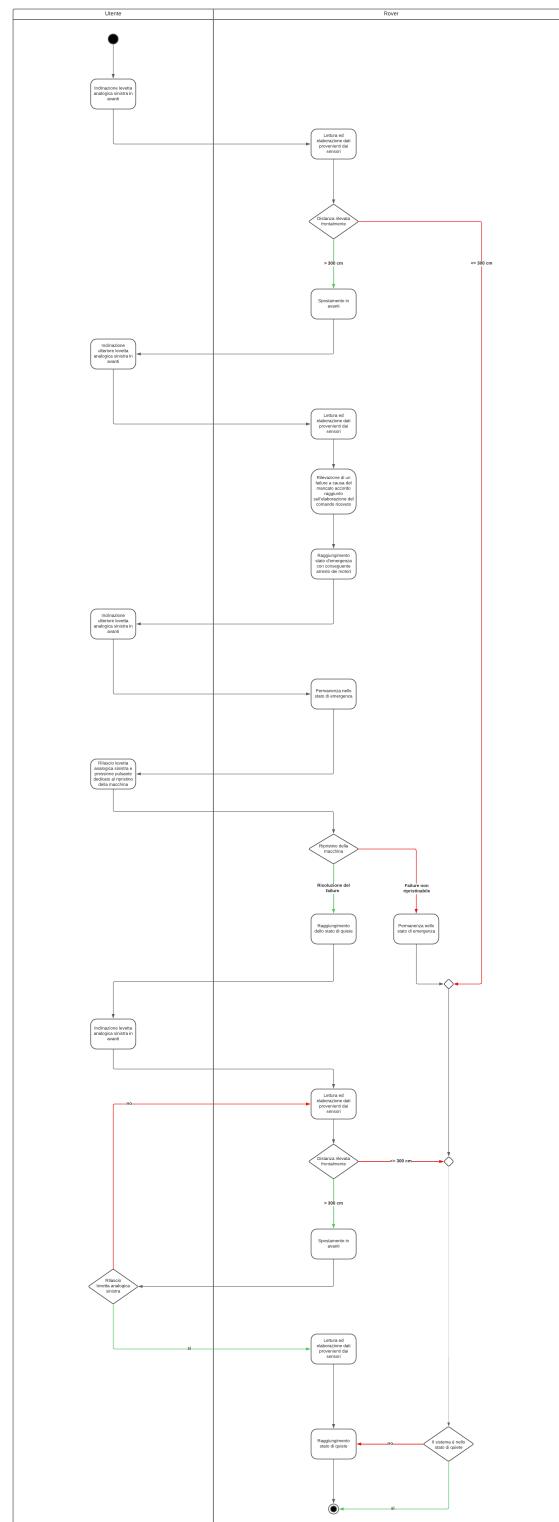


Figura 3.11: Scenario di raggiungimento stato di emergenza del sistema Rover

Capitolo 4

Definizione dello Stato

In questo capitolo tratteremo la definizione dello stato globale del firmware distribuito. È fondamentale definire quali devono essere le variabili di interesse e come queste devono essere interpretate.

4.1 Definizione dello stato

Si riporta nella tabella 4.1 l'insieme di tutte le variabili che compongono lo stato del sistema Rover. Ogni board possiederà la propria versione di tale struttura; i valori in essa contenuti potranno essere aggiornati in parte tramite letture provenienti da sensori a cui la specifica board ha accesso ed in parte da dati che devono necessariamente essere scambiati fra le board tramite un apposito protocollo di comunicazione.

Si riporta inoltre il significato dei diversi bit della variabili control_buttons e control_values nelle tabelle 4.2 e 4.3, in esse sono inserite in maniera sintetica molte informazioni come ad esempio lo stato dei bottoni di interesse associati al controller ma anche l'esito delle operazioni di lettura dai sensori.

Definizione dello Stato

Nome Variabile	Tipo	Dimensione (bit)
motors_direction	uint8_t[4]	8
rpm_motors	float[4]	32
temperature_B1	float	32
temperature_B2	float	32
obstacle_distances	float[3]	32
accX	float	32
gyroZ	float	32
y_analog	int8_t	8
x_analog	int8_t	8
control_buttons	uint8_t	8
control_values	uint8_t	8
braking	BRAKING_TYPE	8
next_state	ROVER_STATE	8
degraded	DEGRADED_STATUS	8
Tot:		440 bit

Figura 4.1: Tabella delle variabili: nome, tipo e dimensione in bit

Definizione dello Stato

Bit	Significato
0	led bianco dx
1	led rosso dx
2	led bianco sx
3	led rosso sx
4	retro speciale
5	emergency stop
6	unused
7	unused

Figura 4.2: Significato dei bit per la variabile `control_buttons`

Bit	Significato
0	accelerometro UP/DOWN
1	joypad UP/DOWN
2-7	unused

Figura 4.3: Significato dei bit per la variabile `control_values`

Sulla base di quanto riportato, lo stato globale avrà un ingombro di XX bit. Andiamo inoltre ad evidenziare quali sono le informazioni inviate dalla board_2 alla board_1 e quali quelle che la board_1 invia alla board_2 riportate rispettivamente in 4.5 e 4.4, questa operazione preliminare sarà fondamentale per il calcolo dei tempi necessari al protocollo di comunicazione fra board per terminare.

Nome Variabile	Tipo	Dimensione (bit)
rpm_motors	float [4]	32
motors_direction	uint8_t [4]	8
temperature_B1	float	32
Tot:		192 bit

Figura 4.4: Tabella delle variabili il cui aggiornamento è sotto la responsabilità della board_1

Definizione dello Stato

Nome Variabile	Tipo	Dimensione (bit)
temperature_B2	float	32
obstacle_distances	float [3]	32
accX	float	32
gyroZ	float	32
y_analog	int8_t	8
x_analog	int8_t	8
control_buttons	int8_t	8
control_values	int8_t	8
Tot:		224 bit

Figura 4.5: Tabella delle variabili aggiornate dalla board_2

A seguito dello scambio dei parametri riportati nelle tabelle 4.4 e 4.5 i device hanno tutti i dati necessari per l'evoluzione della propria macchina a stati; prima che questa divenga effettiva, come chiariremo meglio trattando il protocollo di comunicazione board to board, è necessario che le macchine si scambino la propria intenzione. Quest'ultima è rappresentata dalle variabili riportate nella tabella 4.6.

Nome Variabile	Tipo	Dimensione (bit)
braking	BRAKING_TYPE	8
next_state	ROVER_STATE	8
degraded	DEGRADED_STATUS	8
Tot:		24 bit

Figura 4.6: Tabella delle variabili codificanti l'intenzione di ognuna delle due board

Capitolo 5

Dimensionamento del sistema Rover

Andremo di seguito a stimare il tempo necessario per l'esecuzione delle principali attività del firmware da noi progettato. Analizzando quest'ultimo, è possibile distinguere 3 fasi:

- lettura dei sensori
- scambio delle informazioni proprie di ogni board volto alla creazione di uno stato globale univoco
- attuazione

Ognuna delle fasi sopra descritte richiede lo scambio di informazioni e/o la computazione di dati a partire da informazioni "grezze" lette dai sensori. Tutte le operazioni, siano esse computazioni o interfacciamento con i sensori richiedono del tempo. Il seguente capitolo mira ad approfondire il procedimento teorico che ci ha portato alla stima dei tempi necessari alle diverse operazioni e presenta anche eventuali misurazioni sperimentali a favore di quanto teorizzato.

5.1 Calcolo dei Tempi per le Attività del Firmware

La valutazione del tempo necessario per le attività fondamentali del firmware si articola in tre fasi chiave.

5.1.1 Lettura dei Sensori

La fase di lettura dei sensori costituisce il primo passo nel processo di acquisizione delle informazioni. È cruciale considerare il tempo richiesto per ottenere dati "grezzi" dai diversi sensori. I motivi per cui i tempi di interfacciamento con i sensori possono cambiare sono molteplici e possono derivare da:

- **fattori fisici**, come nel caso del sensore ad ultrasuoni HCSR04, il quale impiega un certo tempo per rilevare ostacoli distanti, correlato alla velocità di propagazione delle onde sonore
- **implementazioni digitali**: sensori con un interfacciamento digitale richiedono l'implementazione di protocolli dedicati, spesso basati su una comunicazione seriale

Andremo ora ad analizzare singolarmente tutti i sensori coinvolti.

HCSR04

Il principio di funzionamento del sensore ad ultrasuoni HCSR04 è il seguente:

- quando sul pin **trig** viene imposto un segnale alto, viene generata un'onda che si propaga nell'ambiente circostante
- se un ostacolo è presente all'interno del range operativo del sensore, l'onda sarà riflessa ritornando al sensore e portando ad un cambio di stato del pin **echo**

Calcolando il tempo che intercorre fra il segnale *trig* e quello *echo* è possibile ottenere la distanza dell'ostacolo.



Figura 5.1: Sensore HCSR04

Supponendo in prima approssimazione che il tempo necessario alla variazione di stato del pin *trig* e per la gestione dell'interrupt associata al cambio di stato del pin *echo* sia nullo, il tempo richiesto per assicurarsi che non ci sia un ostacolo a meno di 3 m dal sensore è dovuto interamente al tempo fisico necessario alla propagazione dell'onda nello spazio. Sulla base di quanto fin ora affermato è possibile costruire un modello analitico del sensore e stimare il tempo necessario per l'ottenimento dell'informazione desiderata:

$$t = 2 \frac{d}{v_{\text{suono}}}$$

dove t è il tempo espresso in s per percorrere una distanza d espressa in m. Il fattore moltiplicativo 2 è necessario per considerare che l'onda deve percorrere la distanza d prima in un verso e poi, per mezzo della riflessione dovuta ad un ostacolo, nell'altro. Per v_{suono} si è utilizzato il valore di 340 m/s , come riportato sulle specifiche del produttore.

Graficando la relazione ottenuta, possiamo ottenere un'idea completa del comportamento del sensore e del tempo impiegato per la rilevazione:

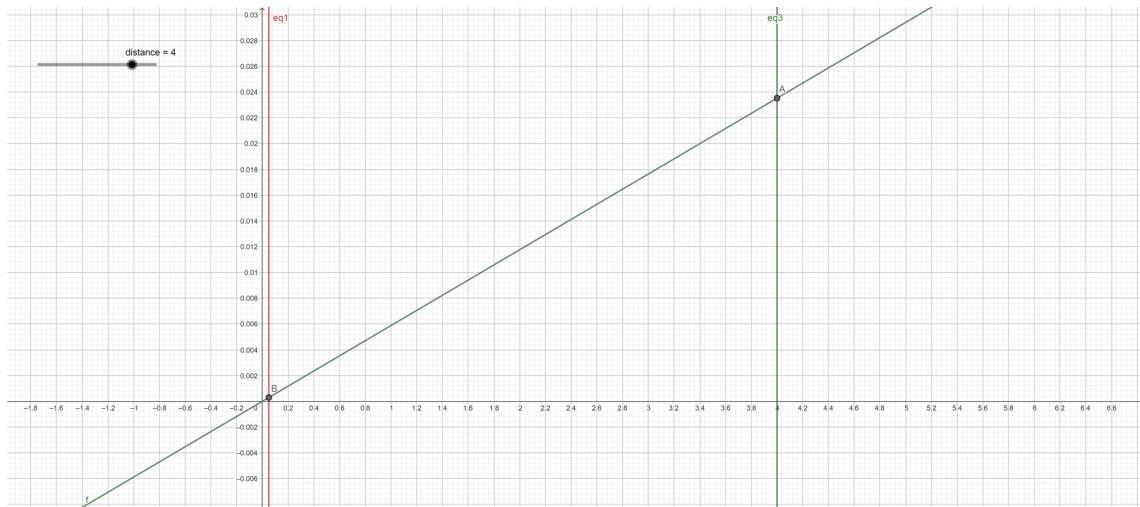


Figura 5.2: Tempi di risposta del sensore HCSR04 al variare della distanza di un ostacolo

A partire dalle misurazioni ottenute tramite questo modello, possiamo supporre che, il tempo necessario ad ottenere un'informazione utile, nel nostro specifico caso applicativo è di 0.0235 s . Inoltre, un ostacolo a 4.5 m di distanza richiederà approssimativamente 0.0264 s per essere individuato.

All'analisi teorica appena presentata, ne aggiungiamo una sperimentale, eseguita sfruttando un pin GPIO tenuto ad un valore logico alto per tutto il tempo necessario alla lettura della distanza di un ostacolo posto a 4.5 m dal sensore. Il tempo impiegato è stato di 0.026 s , in linea con quanto supposto.

PS2 Controller

La comunicazione con il controller PS2 avviene tramite un ricevitore Bluetooth collegato alla *Board_2* mediante il protocollo SPI.

Attenzione

Supponiamo trascurabile il **tempo di propagazione** del segnale radio utilizzato dal protocollo Bluetooth a livello di trasmissione e della propagazione elettrica del segnale SPI all'interno dei jumper che collegano le board. Terremo in considerazione questa approssimazione, aggiungendo un ε di incertezza alla stima teorica ottenuta.

La periferica SPI utilizzata per la comunicazione è stata configurata per ottenere un clock di riferimento sul pin *SCK* con frequenza di 62500 Hz. Considereremo inoltre la velocità di riferimento per il trasferimento dei dati del protocollo *Bluetooth 1.2* pari a 723 kbit/s.

Si è scelto di comunicare con il controller in *analog mode*, che richiede uno scambio periodico di 9 byte di dati in una comunicazione *full duplex*.

Il protocollo SPI, per sua natura, non aggiunge alcun overhead alla comunicazione. Pertanto, i bit inviati sulla linea seriale saranno 72, sia nel collegamento controller - ricevitore *Bluetooth* che in quello *ricevitore Bluetooth - STM32F401RE MainBoard*. Possiamo stimare quindi che, con tale configurazione, sarà necessario un tempo pari a:

- 0.09950 ms per il tratto controller - ricevitore *Bluetooth*
- 1.152 ms per il tratto *ricevitore Bluetooth - STM32*

Pertanto, la comunicazione con il controller richiede circa 1.2515 ms + ε .

All'analisi teorica appena presentata, aggiungiamo una valutazione sperimentale eseguita sfruttando un pin GPIO tenuto ad un valore logico alto per tutto il tempo necessario all'interfacciamento con il controller.

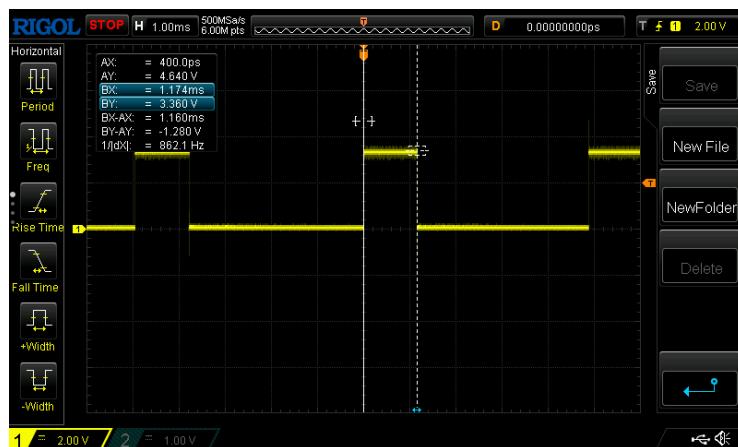


Figura 5.3: Tempi di risposta della funzione *read_gamepad_analog*

Giroscopio/Accelerometro (MPU6050)

Il sensore MPU6050 utilizza il protocollo seriale *I2C* impostato in *fast mode*, capace di trasferire informazioni con una frequenza di 400 kHz.

Nell'interfacciarsi con il sensore, sono necessari i seguenti passi:

- specificare l'indirizzo di memoria del sensore dal quale si desidera leggere, mediante una *I2C write*
- leggere un certo numero di byte dalla memoria del sensore, tramite una *I2C read*

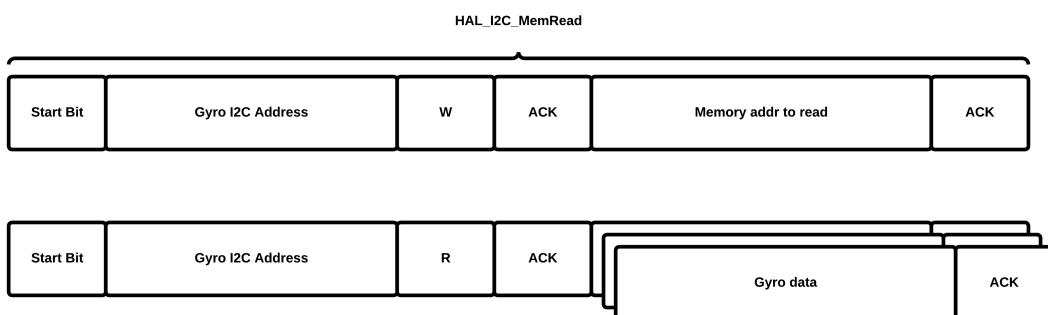


Figura 5.4: Comportamento della `HAL_I2C_MemRead`

L'MPU6050 è in grado di restituire i valori di accelerazione lineare e angolare lungo gli assi X, Y e Z sotto forma di due interi a 8 bit per ogni asse. Pertanto, i byte di dati che devono essere letti sono 6 (due per ogni asse). Il totale di bit che devono essere inviati lungo il canale ammonta pertanto a circa 84.

Nel caso specifico di *I2C*, tuttavia, non è stato possibile ottenere una stima affidabile dei tempi di comunicazione, probabilmente perché, analizzando il normale comportamento di un master in ricezione si ha:

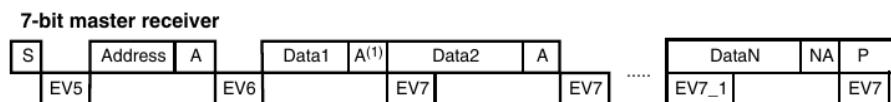


Figura 5.5: "Transfer sequence diagram for master receiver" - STM32F401x reference manual

I diversi eventi EV_n richiedono del tempo che non è semplice stimare per via analitica. Si è quindi optato per un approccio sperimentale, procedendo direttamente alla misurazione del tempo impiegato dalla comunicazione nel caso peggiore tramite oscilloscopio.

Dimensionamento del sistema Rover

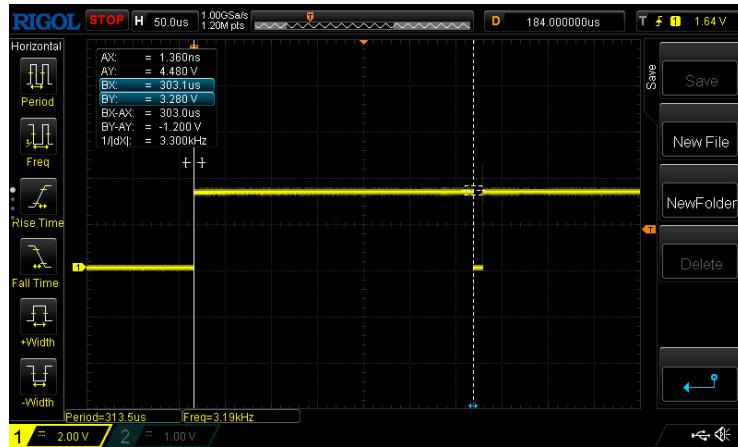


Figura 5.6: Tempi di risposta della funzione `MPU60X0_get_gyro_value`

In figura 5.6 sono riportati i dati ottenuti: si noti che in questo caso, il tempo considerato non include soltanto la comunicazione ma anche l'elaborazione dell'informazione; come precedentemente affermato, infatti, il trasferimento tramite I2C avviene per gruppi di byte, mentre il dato è rappresentato su 16 bit.

Encoder

La lettura degli encoder è gestita tramite un *timer esterno*, il quale calcola il numero di *pulses* registrati in un dato intervallo di tempo tra due rilevazioni consecutive. Basandosi su questa metodologia, si può affermare che il tempo di esecuzione per tale processo è limitato al numero di cicli di clock necessari per eseguire le istruzioni aritmetiche per il calcolo degli RPM. La natura efficiente di questa operazione è confermata dalla misurazione effettuata con un oscilloscopio, che ha cronometrato il tempo richiesto per calcolare la velocità in RPM di tutti e quattro gli encoder. L'immagine ottenuta, come mostrato nella Figura 5.7, evidenzia un impulso alto per una durata di 3.760 μs, che indica il tempo di esecuzione della funzione `encoder_manager_update_rpm()`.

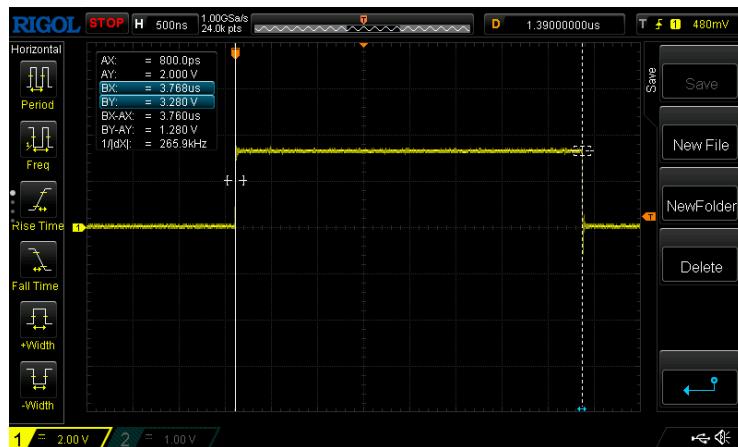


Figura 5.7: Tempi di risposta della funzione `encoder_manager_update_rpm`

5.1.2 Scambio delle Informazioni tra le Board

La comunicazione efficace tra le diverse schede STM32F401RE è cruciale per la creazione di uno stato globale univoco. Nella valutazione di questa fase, si considerano i tempi di trasmissione e ricezione dei dati, oltre alla complessità del protocollo di comunicazione. Il protocollo seriale adottato per la comunicazione fra le board è un elemento chiave in questo contesto.

A causa di limitazioni fisiche che ci hanno costretto a sacrificare numerosi PIN sulla scheda STM32F401RE, non abbiamo potuto sfruttare completamente tutte le sue possibilità. Per la realizzazione della comunicazione *board to board*, si valutato l'utilizzo di I2C e UART. In particolare, analizzeremo di seguito le loro caratteristiche principali per selezionare quello più adatto.

Affidabilità della trasmissione

Il protocollo di comunicazione tra le board rappresenta un momento critico per l'intero sistema, essenziale per la costruzione di uno stato globale condiviso fra i vari dispositivi. Per garantire la correttezza delle informazioni scambiate, utilizziamo codici a ridondanza ciclica (CRC). Questi vengono calcolati e verificati in hardware dalla scheda STM32F401RE. Ad ogni messaggio vengono aggiunti dunque 32 bit, tuttavia, questo introduce un overhead maggiore sulle comunicazioni di minore entità. Si è optato per l'UART principalmente per garantire un livello maggiore di sicurezza nelle comunicazioni, a discapito della velocità di trasmissione. L'I2C offre una modalità sincrona e una frequenza di clock più elevata, la nostra priorità, tuttavia, è stata quella di garantire robustezza ed affidabilità delle trasmissioni, elementi fondamentali per il corretto funzionamento del sistema; pertanto, la scelta è ricaduta su **UART**.

Board to Board protocol

Il protocollo di comunicazione in dettaglio, che nella sua interezza sarà affrontato come riportato in Figura 6.18 durante l'analisi della macchina a stati, può essere riassunto, ad alto livello, come mostrato in Figura 5.8.



Figura 5.8: Vista di alto livello del protocollo di comunicazione fra board

Possiamo distinguere tre diverse macro fasi:

- **scambio dello stato**: come riportato all'interno delle Tabelle 4.4 e 4.5 dovranno essere comunicati in modalità *half-duplex* rispettivamente 192 e 224 bit.
- **valutazione dello stato futuro**: in questa fase non vi sono comunicazioni, tuttavia, ogni board deve eseguire una serie notevole di elaborazioni volte a valutare quale sarà il prossimo stato a partire dalle informazioni ottenute dai sensori. Visto l'ammontare delle operazioni non possiamo escludere questa fase dalle nostre considerazioni
- **accordo sulle proprie intenzioni**: anche in questa fase sarà necessaria una comunicazione; in particolare, il messaggio sarà di pari dimensioni sia per la Board_1 che per la Board_2. Con riferimento alla Tabella 4.6 lo scambio consisterà in un messaggio di 24 bit

La periferica UART utilizzata per la comunicazione è stata configurata con i seguenti parametri:

Baud Rate	115200 simboli/s
Parity	Odd
Data	8

Tabella 5.1: Setting della periferica UART utilizzata per la comunicazione fra board

Sulla base dei parametri forniti, possiamo stimare il tempo necessario all'invio delle informazioni lungo il canale:

- **scambio dello stato**

- **Board_1 → Board_2:** 192 bit payload + 32 bit CRC + 28 bit (parità) + 28 start bit + 28 stop bit, per un totale di 308 bit; saranno dunque necessari circa 2.67 ms
- **Board_2 → Board_1:** 224 bit payload + 32 bit CRC + 32 bit (parità) + 32 start bit + 32 stop bit, per un totale di 352 bit; saranno dunque necessari circa 3.05 ms
- **valutazione dello stato futuro:** in questa fase sono eseguiti un insieme di controlli sui dati che possono essere ricondotti ad una serie in cascata di if; pertanto, terremo in considerazione un tempo ε fissato a circa 2 ms
- **scambio dell'intention:** 24 bit payload + 32 bit CRC + 7 bit (parità) + 7 start bit + 7 stop bit, per un totale di 64 bit; saranno dunque necessari 0.66 ms

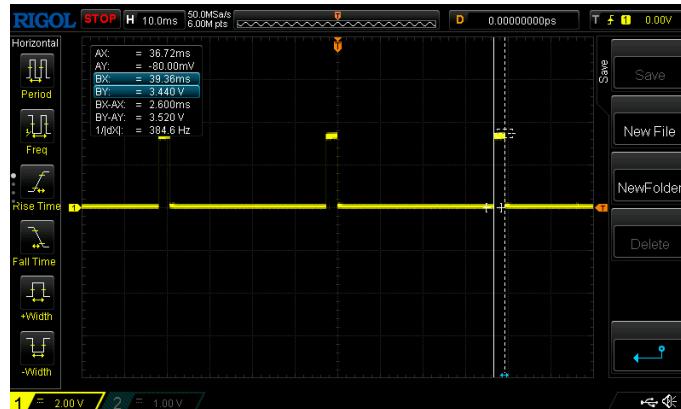


Figura 5.9: Trasmissione dello stato **Board_1 → Board_2** misurata: 2.6 ms

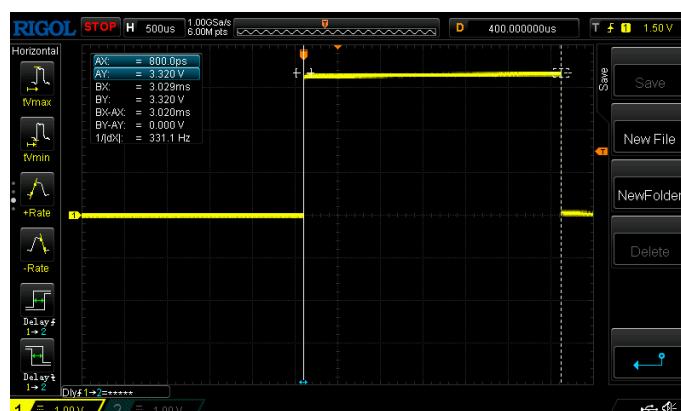


Figura 5.10: Trasmissione dello stato **Board_2 → Board_1** misurata: 3.02 ms

Analizziamo ora il protocollo con maggiore dettaglio tramite il seguente sequence diagram

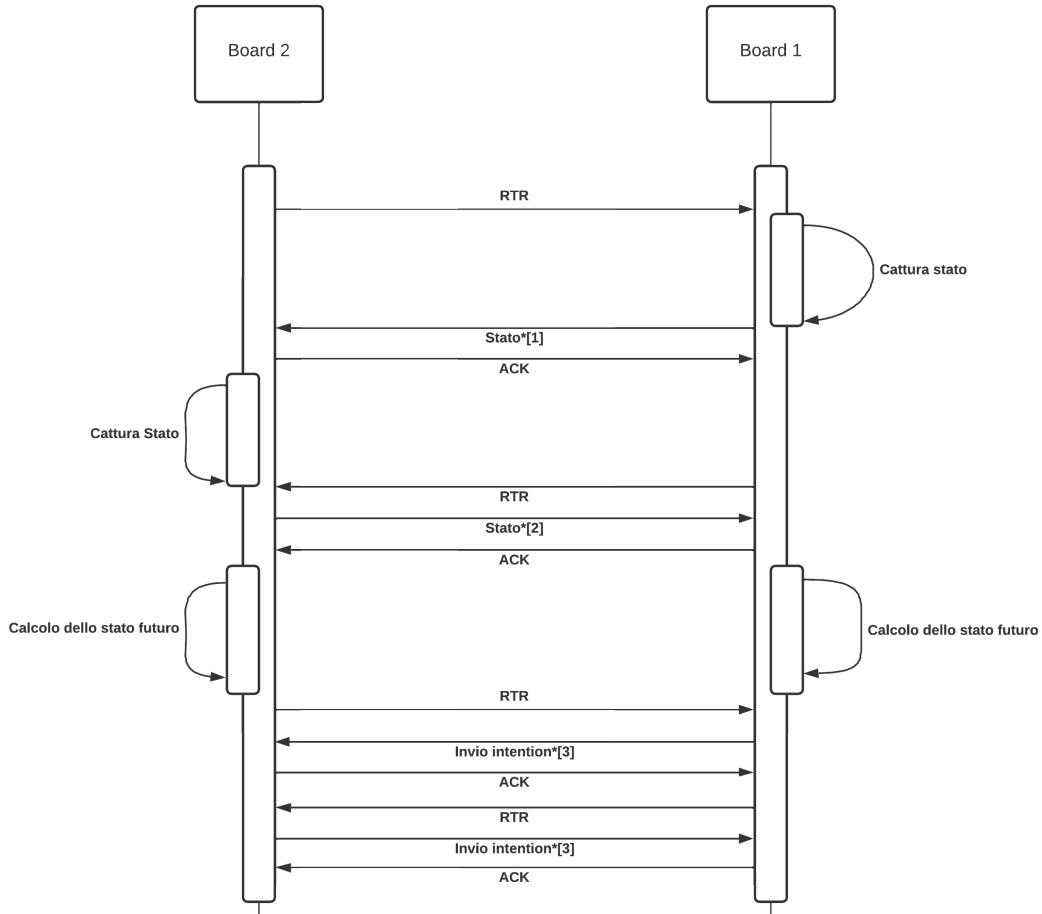


Figura 5.11: Protocollo Board to Board.

Per quanto riguarda i dettagli sullo stato e le intention indicati all'interno della Figura 5.11, nel caso:

- **Stato*[1]** si faccia riferimento alla Tabella 4.4
- **Stato*[2]** si faccia riferimento alla Tabella 4.5
- **Intention*[3]** si faccia riferimento alla Tabella 4.6

Nella stima del tempo totale impiegato dal protocollo per essere eseguito, dobbiamo considerare che:

1. RTR ed ACK sono stati pensati come segnali hardware e saranno implementati tramite GPIO, pertanto possiamo supporre il tempo necessario all'invio di questi ultimi trascurabile
2. per garantire robustezza, l'invio dello stato e delle intention può essere rieseguita al più una volta laddove non dovesse pervenire alcun ACK per

segnalare l'avvenuta ricezione entro un tempo limite, con tali tempi indicati all'interno della macchina a stati con la dicitura "ALIVE_TIMEOUT_XX"

L'invio degli stati da parte delle board e le intention sono stati invece calcolati precedentemente. L'analisi che segue tiene conto del $WCET_{est}$, ovvero del *worst case execution time* stimato. In tale scenario dobbiamo supporre che ogni volta che una board invia un messaggio, questo fallisca inizialmente, per poi riuscire al secondo re-invio.

Azione	Tempo impiegato
Invio Stato Board_1	2×2.67 ms
ALIVE_TIMEOUT_BOARD_1_STATE	2.8 ms
Invio Stato board_2	2×3.05 ms
ALIVE_TIMEOUT_BOARD_2_STATE	3.2 ms
Calcolo stato futuro	2 ms
Invio intention	4×0.66 ms
ALIVE_TIMEOUT_INTENTION	2×0.8 ms
Tot:	23.68 ms

Tabella 5.2: Tempo impiegato dal task di comunicazione per la costruzione di uno stato globale univoco

Una versione più efficiente del protocollo potrebbe essere realizzata supponendo una comunicazione UART robusta al punto da commettere al massimo un errore su un singolo messaggio. Nel caso peggiore, questo errore si verificherebbe sul messaggio che richiede più tempo per essere inviato, portando a quanto espresso nella Tabella 5.3.

Azione	Tempo impiegato
Invio Stato board_1	2.67 ms
Invio Stato board_2	2×3.05 ms
ALIVE_TIMEOUT_BOARD_2_STATE	3.2 ms
Calcolo stato futuro	2 ms
Invio intention	2×0.66 ms
Tot:	14.63 ms

Tabella 5.3: Tempo impiegato dal task di comunicazione per la costruzione di uno stato globale univoco nel caso in cui vi sia un'unica ritrasmissione complessiva

5.1.3 Attuazione

Il controllo indipendente dei 4 motori a disposizione del sistema Rover avviene tramite l'interfacciamento via protocollo seriale UART con 2 motor driver Sabertooth2X12. Tali device permettono l'interfacciamento via UART con i seguenti parametri:

Baud Rate	9600 simboli/s
Parity	No
Data	8 bit

Tabella 5.4: Setting della periferica UART utilizzata per la comunicazione con i driver Sabertooth2x12

Ogni comando inviato ai driver richiede un pacchetto strutturato su 4 byte e permette il controllo di un singolo motore. Qualsiasi azione, sulla base di quanto mostrato fino ad ora richiede il controllo di ogni motore; devono essere dunque trasferiti 16 byte articolati in 4 comunicazioni indipendenti. Lungo il canale di comunicazione saranno trasferiti circa 160 bit impiegando circa 16.6 ms come riportato in Figura 5.12

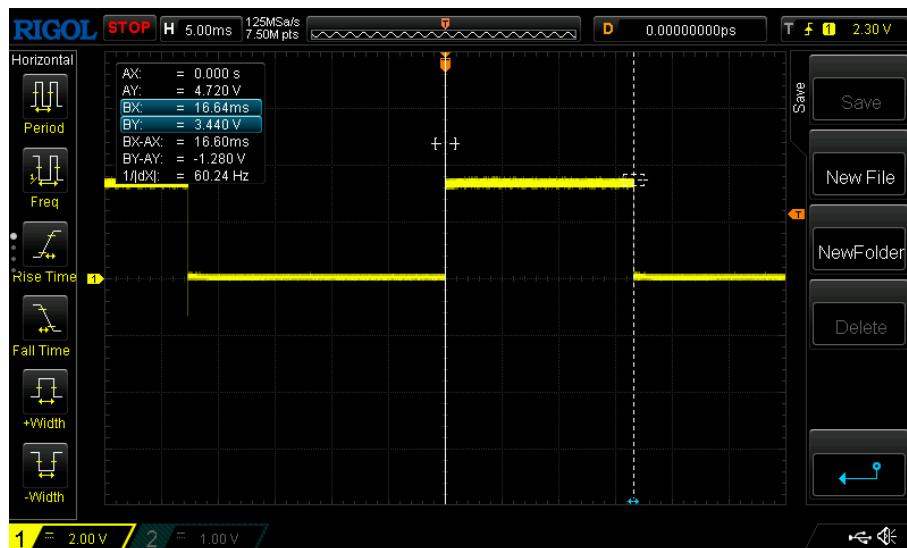


Figura 5.12: Tempo impiegato: 16.64 ms

Il sistema Rover è inoltre munito di una coppia di fari che possono essere comandati dalla *Board_1* e pertanto l'attuazione in questo caso consiste nell'andare ad impostare ad un valore logico alto un pin, avendo dunque un tempo impiegato trascurabile.

5.2 Progettazione dei task

La filosofia alla base della progettazione dei task cerca di minimizzarne il numero. L'uso di tale approccio, congiunto alle politiche di scheduling adottate, ha consentito al sistema Rover un efficace:

1. **minimizzazione delle Dipendenze:** si è posta particolare attenzione a ridurre al minimo le dipendenze fra i vari task all'interno di ciascuna board. Questo approccio mira a migliorare la modularità del sistema, consentendo una maggiore flessibilità nella gestione e nell'aggiornamento dei singoli task senza impattare pesantemente sugli altri
2. **mapping del Ciclo di Controllo:** i task sono stati progettati considerando il ciclo naturale di controllo del rover, suddividendolo in tre fasi chiave: *leggi, elabora, esegui*. Questa suddivisione consente una gestione più chiara delle attività del rover, facilitando il monitoraggio e l'ottimizzazione delle prestazioni
3. **sincronizzazione inter-board:** essenziale per un funzionamento coordinato delle due STM32 che costituiscono il sistema, assicura che le operazioni siano sincronizzate attraverso le politiche di scheduling, con una valutazione dettagliata prevista nelle fasi successive del progetto
4. **sincronizzazione intra-board e col mondo fisico:** acquisendo i dati dai sensori in istanti temporalmente molto vicini tra loro ci si assicura una coerenza fisica dei dati acquisiti, specialmente per sensori diversi ma con grandezze fisiche misurate correlate (ad esempio, si desidera che le acquisizioni dei tre sonar avvengano in istanti di tempo quanto più vicini possibile)

Andremo ora ad affrontare con un maggiore livello di dettaglio i diversi punti fin ora trattati.

5.2.1 Minimizzazione delle Dipendenze

Abbiamo esaminato come ciascuna board possieda uno stato proprio, rappresentato nelle Tabelle 4.4 e 4.5 ed una vista dello stato globale, data dall'unione degli stati delle due board rappresentato nella Tabella 4.1. La fase di lettura/scrittura delle proprie variabili di stato avviene in due momenti naturalmente concorrenti:

- durante la lettura dei sensori, il contenuto delle variabili che possono essere aggiornate dalla specifica board viene aggiornato
- durante l'esecuzione del protocollo *board to board*, tali informazioni vengono lette, scambiate ed utilizzate per costruire un'unica vista dello stato globale del sistema, utilizzata come base per l'elaborazione di una *intention*, strutturata come definito nella Tabella 4.6

L'analisi sottolinea la necessità di una struttura dati condivisa, accessibile sia durante la lettura dei sensori, sia durante l'esecuzione del protocollo di comunicazione, contenente l'intera vista globale del sistema.

Un dettaglio cruciale è garantire, in ogni momento, una cattura coerente della realtà all'interno della struttura dati contenente lo stato; tale struttura dati deve per questo essere sempre consistente. Ciò implica che durante l'esecuzione del protocollo *board to board* non è consentito alcun aggiornamento dello stato.

5.2.2 Mapping del ciclo di controllo e sincronizzazione intra-board

Ragionando sul funzionamento previsto per il sistema è possibile individuare tre momenti logicamente separati:

1. lettura delle informazioni dai sensori
2. elaborazione di tali informazioni coronata da una decisione circa il nuovo possibile stato del sistema
3. attuazione sulla base della precedente decisione

Si è deciso di trasportare tale separazione anche nella progettazione dei task eseguendone un mapping 1:1. Durante il normale funzionamento del sistema avremo dunque in esecuzione 3 task con esigenze real time: il primo volto alla lettura dei sensori, il secondo volto all'esecuzione del protocollo *board to board* per l'aggiornamento dello stato globale del sistema ed il terzo volto all'attuazione dell'*intention* comunemente accordata.

Idea

Si è deciso di accoppare in un unico task la lettura di tutti i sensori per garantire l'acquisizione di un'informazione comune ad ognuno di essi che sia nel tempo il più vicino possibile.

Periodo dei task

Abbiamo visto come il task di lettura dei sensori consenta a ciascuna board di acquisire parte delle informazioni necessarie per l'aggiornamento dello stato globale. Tuttavia, affinché lo stato evolva in comune accordo, è essenziale che entrambe le board condividano gli stessi dati. Inoltre, il task di attuazione, che modificherà lo stato fisico del sistema, deve portarlo in uno stato che sia in conformità con questa visione globale. Di conseguenza, se il periodo del task di lettura dei sensori fosse troppo breve rispetto a quello di comunicazione non si otterrebbe alcun beneficio vista la necessità di mantenere coerenza nei dati condivisi.

Tale osservazione può essere resa ancor più evidente facendo riferimento alla situazione in figura:

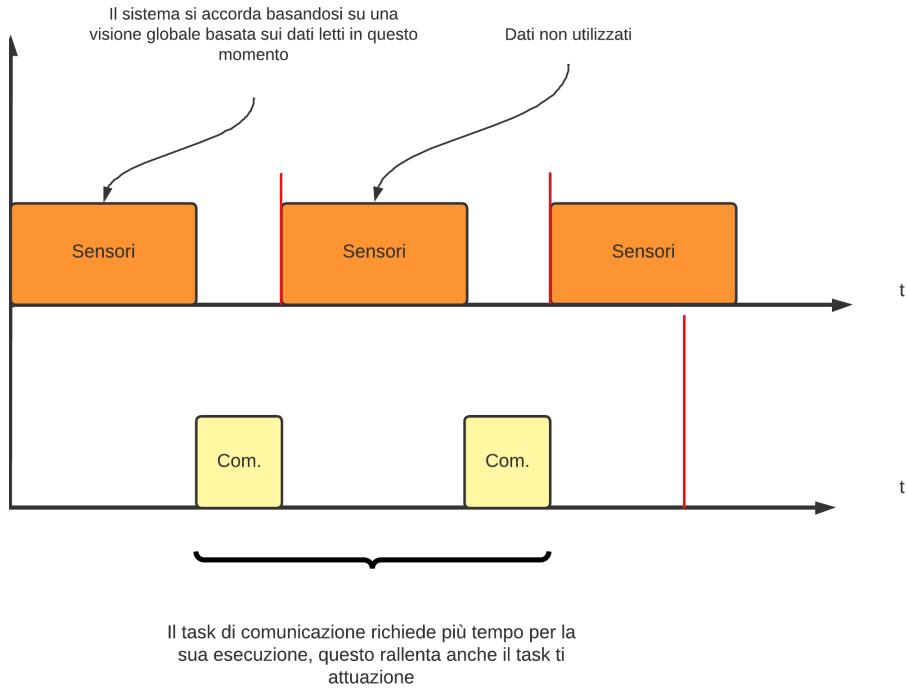


Figura 5.13: Esempio di scheduling dove il task di lettura dei sensori ha un periodo sensibilmente inferiore rispetto a quello di comunicazione.

Per prevenire la problematica evidenziata, si è scelto di **mantenere lo stesso periodo tra il task di lettura dei sensori e quello di comunicazione**. In questo modo, ogni lettura dei sensori sarà comunicata e quindi utilizzata per la successiva evoluzione dello stato, massimizzando così l'efficienza complessiva del sistema.

Un ulteriore problematica evidenziata in Figura 5.13 riguarda l'implicita relazione che intercorre fra il task di attuazione e quello di comunicazione. Il task di attuazione **deve** sempre occuparsi di portare il sistema fisico in accordo con l'*intention* costruita a partire dall'ultima visione globale del sistema concordata. Anche in questo caso, non avrebbe senso mantenere un periodo più basso rispetto al task di comunicazione e di lettura dei sensori in quanto il comando da impartire ai driver sarebbe sempre lo stesso.

5.2.3 Sincronizzazione inter-board

Periodicamente, durante l'esecuzione del task di comunicazione, le board si scambiano le informazioni necessarie alla costruzione dello stato globale attraverso il protocollo illustrato nella Figura 5.11.

Attenzione

Affinché lo scambio vada a buon fine, è necessario che entrambe le board eseguano il task di comunicazione contemporaneamente.

Sebbene sia possibile supporre che le schede passino nello stato di *working* contemporaneamente, la durata del task di lettura dei sensori delle due board può differire nel caso peggiore. In particolare, la Board_1 potrebbe eseguire il task di lettura dei sensori nel caso peggiore in un tempo minore rispetto alla Board_2, e può da subito iniziare il task di comunicazione rimanendo in attesa attiva dell'RTR. Questa configurazione consente alla Board_2 di iniziare il protocollo ricevendo immediatamente una risposta; ciò agevola lo scambio e la sincronizzazione dello stato globale.

5.3 Scelta dell'algoritmo di scheduling

5.3.1 Dimostrazione di schedulabilità

A valle delle considerazioni esposte e della necessità di assegnare un "ordine" all'esecuzione dei task, la scelta migliore circa l'algoritmo di scheduling sembrerebbe ricadere su **Timeline scheduling**, anche detto **Cyclic Executive**. Il metodo consiste nel dividere l'asse temporale in slot di uguale lunghezza, in cui uno o più task possono essere allocati per l'esecuzione, in modo tale da rispettare le frequenze derivate dai requisiti applicativi.

Per quanto concerne la Board_2, il costo computazionale dei task in esecuzione è così definito:

- **Lettura dei sensori**
 - **Lettura controller**: 1.3 ms
 - **Lettura Accelerometro**: 0.30 ms
 - **Lettura HCSR04**: 24 ms
- **Comunicazione**
 - in questo caso tutte le attività del task coincidono con l'esecuzione del protocollo stimato nella Tabella 5.2, ovvero 24 ms
- **Attuazione**
 - **comunicazione con la scheda Sabertooth2X12**: 17 ms

In merito alla scelta del periodo, come esposto nei paragrafi precedenti, lo si è posto pari a 70 ms, in modo da consentire un'esecuzione "sequenziale" dei tre task.

Le informazioni precedentemente riportate sono di seguito schematizzate:

	C	T
Sensori	26	70
Comunicazione	24	70
Attuazione	17	70

Tabella 5.5: Definizione dei task, la durata è espressa in *ms*

Se C_A, C_B e C_C indicano i tempi di esecuzione dei task, è sufficiente verificare che

$$C_A + C_B + C_C \leq 70ms \quad (5.1)$$

Per quanto concerne la Board_1, il costo computazionale dei task in esecuzione è così definito:

- **Lettura dei sensori**

- **Lettura encoder:** la procedura di acquisizione dei dati dai quattro encoder si completa in un tempo massimo di $4\mu s$. Questo intervallo temporale è classificato come deterministico a causa dell'omogeneità delle operazioni aritmetiche impiegate nel calcolo degli RPM. Il tempo di esecuzione è invariante rispetto alla quantità di pulses registrati dal timer hardware in un determinato lasso di tempo. Tale costante di tempo è influenzata esclusivamente dal numero di encoder, il quale, nel contesto applicativo specifico, rimane costante

- **Comunicazione**

- il tempo impiegato dal task di comunicazione è almeno pari a quello impiegato per l'esecuzione del protocollo di comunicazione a cui va aggiunto il tempo in cui la Board_1 è in attesa dell'RTR della Board_2, che nel caso peggiore è dato dalla differenza fra il task di lettura dei sensori della Board_2 e quello della Board_1, ovvero 25 ms. Pertanto, supponiamo che il tempo totale sia di circa 49 ms

- **Attuazione**

- **Accensione/Spegnimento led:** 1 ms

Nota

Per quanto riguarda la board_1 le attività del task di lettura dei sensori e del task di attuazione impiegano un tempo inferiore ad 1 ms, tuttavia, il costo più piccolo possibile assegnabile ad un task all'interno dello scheduler a nostra disposizione è proprio pari ad 1 ms, pertanto si è preso in considerazione tale tempo.

Le informazioni precedentemente riportate sono di seguito schematizzate:

	C	T
Sensori	1	70
Comunicazione	49	70
Attuazione	1	70

Tabella 5.6: Definizione dei task, la durata è espressa in *ms*

Dimensionamento del sistema Rover

Se C_A, C_B e C_C indicano i tempi di esecuzione dei task, è sufficiente verificare che

$$C_A + C_B + C_C \leq 70ms \quad (5.2)$$

Di seguito viene riportata una rappresentazione dello scheduling previsto sia inter-board che intra-board.

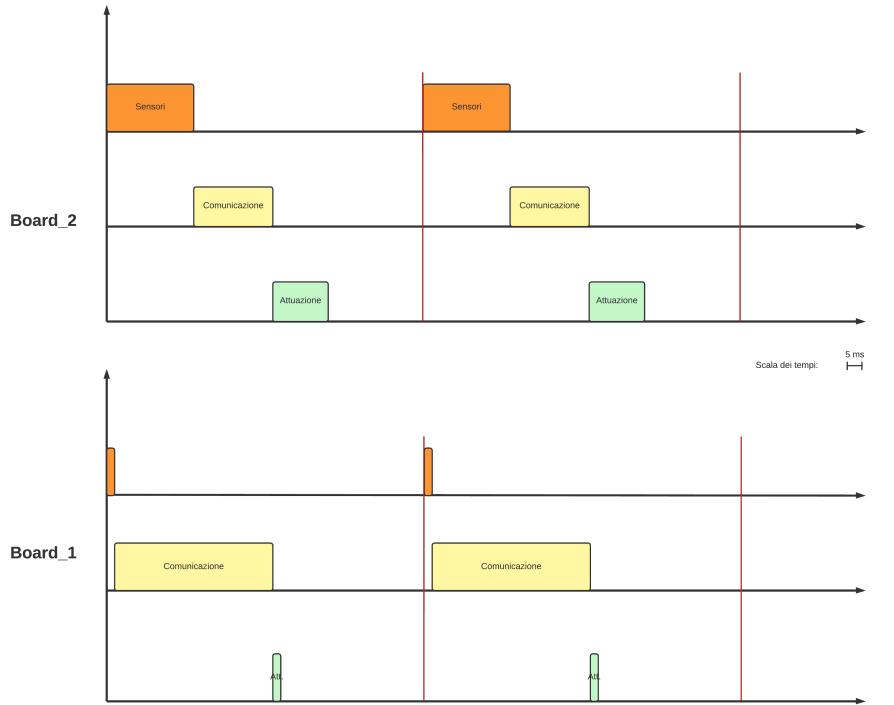


Figura 5.14: Scheduling previsto

Funzionamento degradato

Laddove dovesse verificarsi un fallimento della **Board_1**, come discusso nel Capitolo 2, è previsto che il sistema si sposti in una modalità di funzionamento degradato. In un contesto single board non ha più senso parlare di *task di comunicazione* in quanto il sistema non è più da considerarsi distribuito. Al suo posto terremo in considerazione un ulteriore task, in cui l'unica operazione effettuata è quella di costruire lo stato futuro a partire dai dati campionati dal task di lettura dei sensori della singola **Board_2**. Terremo in considerazione la stessa durata stimata nel caso precedente per l'elaborazione dello stato ovvero 2 ms. Pertanto, il nuovo task set periodico sarà:

	C	T
Sensori	26	70
Calcolo stato futuro	2	70
Attuazione	17	70

Tabella 5.7: Definizione dei task, la durata è espressa in *ms*

Nuovamente, se C_A, C_B e C_C indicano i tempi di esecuzione dei task, è sufficiente verificare che

$$C_A + C_B + C_C \leq 70\text{ms} \quad (5.3)$$

5.4 Spazio minimo di frenata

Ora che il problema è stato formalmente definito, abbiamo tutti i dati necessari a stimare, in base a quanto progettato fin ora, quale sarà lo spazio minimo di frenata del sistema. Lo scenario peggiore in cui il sistema può trovarsi è il seguente:

1. il rover si sta spostando in avanti alla massima velocità (3 m/s)
2. il task di lettura dei sensori ha appena campionato lo stato del mondo fisico (task di lettura dei sensori eseguito) e **non sono presenti ostacoli**
3. un ostacolo di colpo si presenta all'interno del range di frenata critica $distanza \leq 70\text{ cm}$

Prima che il sistema possa effettuare la frenata è necessario attendere:

- 60 ms, affinché il task di lettura dei sensori sia nuovamente portato a termine
- 24 ms affinché l'informazione **ostacolo ad una distanza minore di 70 cm** venga propagata dalla Board_2, che ha effettuato la lettura, alla Board_1

Il tempo di attuazione in questo caso è supposto nullo in quanto la frenata di emergenza viene realizzata direttamente mettendo a ground un pin sulla scheda Sabertooth2X12 dal task di Emergenza. In questo scenario sono stati impiegati 94 ms ad una velocità di 3 m/s, percorrendo uno spazio di **28.2 cm**

Nota

La situazione descritta utilizza i tempi nel caso peggiore di tutti i task. Il task di lettura dei sensori, tuttavia, vista la bassa distanza dell'ostacolo non impiegherà sicuramente 26 ms, ma soltanto una piccola frazione, essendo che nel calcolo del $WCET_{est}$ di tale task gioca un ruolo fondamentale la massima distanza rilevabile dai sensori ad ultrasuoni, molto superiore a 70 cm. Resta invece valida l'assunzione sul caso peggiore del task di comunicazione in quanto potrebbero sempre verificarsi errori dovuti alla trasmissione dello stato.

Pertanto possiamo assumere, in via conservativa, che la distanza minima da un ostacolo per il quale è possibile frenare in sicurezza, nel caso del nostro sistema è di circa **30 cm**.

5.4.1 Un problema implementativo

Nel capitolo precedente è stato sottolineato come fosse un presupposto fondamentale per il corretto funzionamento del protocollo l'esecuzione del task di comunicazione contemporanea su entrambe le board. In fase progettuale, il problema è stato affrontato costruendo lo scheduling in Figura 5.14; tuttavia, ciò si basa su un'altra assunzione implicita, ovvero che entrambe le board siano sincronizzate rispetto a un riferimento temporale esterno. Questa assunzione隐式 diventa ancora più evidente se si tiene conto che le bande verticali rosse in Figura 5.14 indicano che le deadline cadono **sempre contemporaneamente**.

Data la natura puramente distribuita del sistema, tuttavia, non è possibile considerare vera un'assunzione del genere. Infatti, se si considerano due clock identici che oscillano alla stessa frequenza, questi saranno soggetti a un fenomeno noto come "*clock drifting*", il cui impatto è caratterizzato in Figura 5.15.

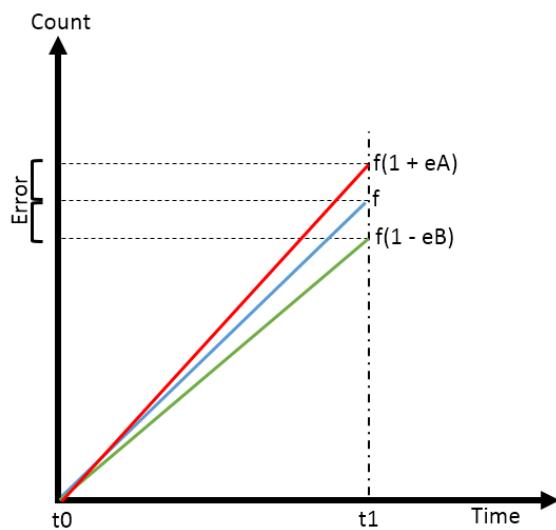


Figura 5.15: Effetti del *clock drifting* su due clock identici con la stessa frequenza di oscillazione

Sebbene i periodi dei task vengano rispettati in accordo alle *deadline hard real time* identificate, non possiamo assumere che rispetto a un terzo riferimento temporale queste cadano contemporaneamente. Tale problema, individuato a seguito dell'implementazione del protocollo, ci ha portato a rivederne il funzionamento.

Per far fronte alla problematica evidenziata si è deciso di rendere evidente il ruolo della board_2 come Master e renderla l'unica a possedere un task di comunicazione, in particolare, nella versione modificata del protocollo l'invio

di un segnale hardware di *RTR* (*ready to receive*) innescata una serie di interruzioni tramite cui i passi del protocollo descritto in figura 5.11 l'intero protocollo. Tale approccio non dipende più da un'implicita sincronizzazione temporale, ma rende esplicita l'esecuzione dell'intero protocollo.

Attenzione

La board_1, in quanto slave, sarà costretta ad eseguire il protocollo in ogni momento su richiesta della board master. Per evitare che possano esserci momenti in cui la comunicazione cada durante l'aggiornamento dello stato da parte della board_1 l'aggiornamento dello stato sarà eseguito in modo atomico, interrompendo le interruzioni in tale fase.

Garanzie hard realtime nel nuovo protocollo

Nel protocollo originariamente progettato, grazie alla presenza di un task periodico di comunicazione in ambedue le board, il rispetto dei vincoli real time poteva essere verificato e garantito dalla presenza dello schedulatore. Il nuovo protocollo, tuttavia, sebbene venga avviato dal task di comunicazione periodico della board_2, viene eseguito interamente per mezzo di una sequenza di callback associate ad interruzioni ed è per questo complesso garantire vincoli real time tramite lo schedulatore.

Tenendo bene a mente quelle che sono le necessità real time del sistema in sviluppo, ciò di cui abbiamo realmente bisogno è un protocollo in grado di terminare e dunque giungere alla costruzione di un nuovo stato, entro una certa deadline sufficiente a garantire una buona reattività ai comandi e al contempo le garanzie di resilienza da attribuire ad un sistema distribuito.

Si è pertanto deciso di rendere il task di comunicazione periodico schedulato dalla board_2 in grado di capire se il protocollo iniziato al periodo precedente sia o meno terminato, nel caso questo sia ancora in corso, possiamo considerare tale situazione al pari di una deadline sforata, il sistema pertanto potrebbe essere in pericolo in quanto il proprio stato è diventato il campionamento di un istante troppo distante nel tempo da quello attuale, introducendo un rischio per il sistema. In tali situazioni, la board_2 interromperà la comunicazione per proseguire le proprie attività in modalità degradata.

5.4.2 Scheduling finale

A seguito di quanto discusso fino a questo momento, si è pervenuti allo scheduling seguente:

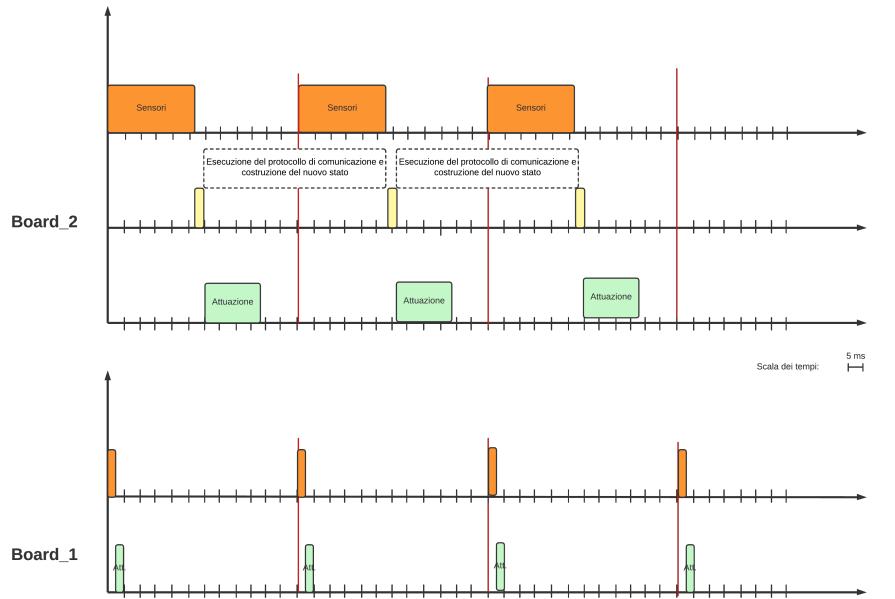


Figura 5.16: Scheduling finale ottenuto

Si noti che, il periodo dei task su entrambe le board è stato abbassato a 60 ms questo perché il task di comunicazione nella board_1 è stato portato alla durata, nel caso peggiore, di 1 ms, in quanto ora ha soltanto la funzione di verificare la terminazione del protocollo e in caso affermativo avviare una nuova esecuzione.

La scelta di utilizzare per il protocollo trasmissioni non bloccanti ha reso inoltre possibile portare avanti il protocollo di comunicazione anche mentre sono in esecuzione trasmissioni e/o ricezioni, l'impatto sul task di attuazione in entrambe le board è pertanto ridotto al minimo.

Attenzione

A seguito del nuovo protocollo l'attuazione potrebbe essere ritardata di un periodo. Tale sacrificio sulle prestazioni è stato necessario affinché fosse possibile risolvere

5.4.3 Una soluzione alternativa al nuovo protocollo presentato

Il protocollo inizialmente progettato e presentato nel corso dei precedenti capitoli può essere utilizzato in maniera coerente a quanto descritto supponendo di eliminare il problema del *clock-drifting*, ciò può essere ottenuto tramite l'utilizzo di un clock esterno in grado di fornire un unico riferimento temporale ad entrambi i dispositivi.

Capitolo 6

Macchina a stati

Il seguente capitolo si concentra sulla modellazione del sistema attraverso l'utilizzo di *Stateflow* nell'ambiente Matlab. In particolare, la macchina a stati che modella il firmware distribuito è stata realizzata anche con l'obiettivo di semplificare l'implementazione attraverso la generazione automatica del codice attraverso un ulteriore strumento, l'*Embedded Coder*.

6.1 Approccio Generale

L'intera macchina è racchiusa in un unico grande chart in decomposizione parallela denominato *Rover*, all'interno del quale troviamo due macrostati, *Board_1* e *Board_2*, che modellano il concetto di *ridondanza* e distribuzione hardware/software.

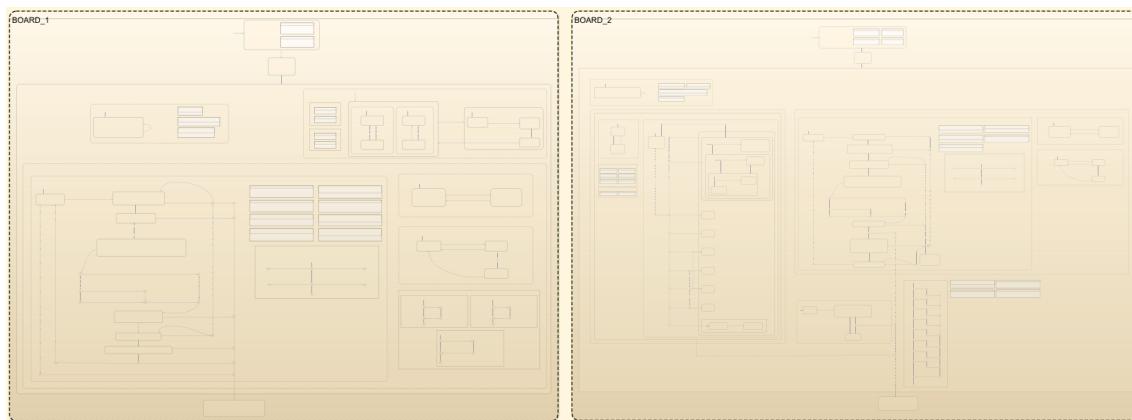


Figura 6.1: Rover chart

A questo livello, sono definite tutte le variabili che devono essere condivise tra le due board e sono riassunte nella seguente tabella.

Tabella 6.1: Variabili globali del chart

Name	Type	Value	Description
------	------	-------	-------------

Macchina a stati

READ_PERIOD	double	10 msec	Periodo di lettura e aggiornamento delle variabili associate ai sensori
COMM_PERIOD	double	12 msec	Periodo con cui deve avvenire la comunicazione tra le due board
COMM_TIMEOUT	double	3 msec	Attesa prima di considerare un fallimento nella comunicazione
B2_ALIVE	double	15 msec	Attesa prima di considerare un failure nella BOARD_2
DISTANT_OBSTACLE_DISTANCE	double	3 m	Distanza minima ostacolo 'lontano'
NEARBY_OBSTACLE_DISTANCE	double	0.7 m	Distanza massima ostacolo 'improvviso'
MIN_NORMAL_TEMP	double	0	Temperatura operativa minima
MAX_NORMAL_TEMP	double	35	Temperatura operativa massima
CRITICAL_TEMP	double	50	Temperatura critica
TEMP_TIMEOUT	double	15 sec	Attesa prima di entrare in modalità degradata nel caso la temperatura superi la MAX_NORMAL_TEMP
JOYPAD_MASK	uint8	2	Maschera utilizzata per verificare il corretto funzionamento del joypad
MPU_MASK	uint8	1	Maschera utilizzata per verificare il corretto funzionamento dell'accelerometro/giroscopio
ALT_BACKWARD_MASK	uint8	24	Maschera utilizzata per verificare l'attivazione della combo e del comando per attivare la retromarcia alternativa
MIN_SIDE_DISTANCE	double	0.1 m	Distanza minima per ruotare sul baricentro del Rover
B1_state	[double]		Array utilizzato per simulare lo scambio dello stato tra le board relativo alla BOARD_1
B2_state	[double]		Array utilizzato per simulare lo scambio dello stato tra le board relativo alla BOARD_2
B1_intention	[double]		Array utilizzato per simulare lo scambio dell'intenzione tra le board relativa alla BOARD_1
B2_intention	[double]		Array utilizzato per simulare lo scambio dell'intenzione tra le board relativa alla BOARD_2
B1_RTR	Event		Segnala che la BOARD_1 è pronta a ricevere
B2_RTR	Event		Segnala che la BOARD_2 è pronta a ricevere
RESET	Event		Segnala che il rispettivo RTR è stato preso in carico

B1_ACK	Event	Segnala che la BOARD_1 ha ricevuto correttamente le informazioni
B2_ACK	Event	Segnala che la BOARD_2 ha ricevuto correttamente le informazioni
START_TEMP_TIMER	Event	Segnala che deve essere avviato il timer relativo al controllo della temperatura
RESET_TEMP_TIMER	Event	Segnala che il timer relativo al controllo della temperatura può essere ripristinato

Idea

Tramite la decomposizione presentata, si vuole modellare il sistema *Rover* come un sistema distribuito le cui operazioni dipendono dallo stato in cui versano le macchine a stati implementate dalle due singole board.

6.2 BOARD_1

Passiamo quindi alla descrizione della *Board_1* che, nel caso in esame, svolge la funzione di una sorta di "slave", dal momento in cui ha il compito di ottenere i dati dai sensori a essa connessi e aggiornare le proprie variabili di stato, comunicarle alla *BOARD_2* quando richiesto e gestire la parte di attuazione riguardante l'accensione e lo spegnimento dei led. Una panoramica dell'intera *BOARD_1* è mostrata nella figura 6.2.

Macchina a stati

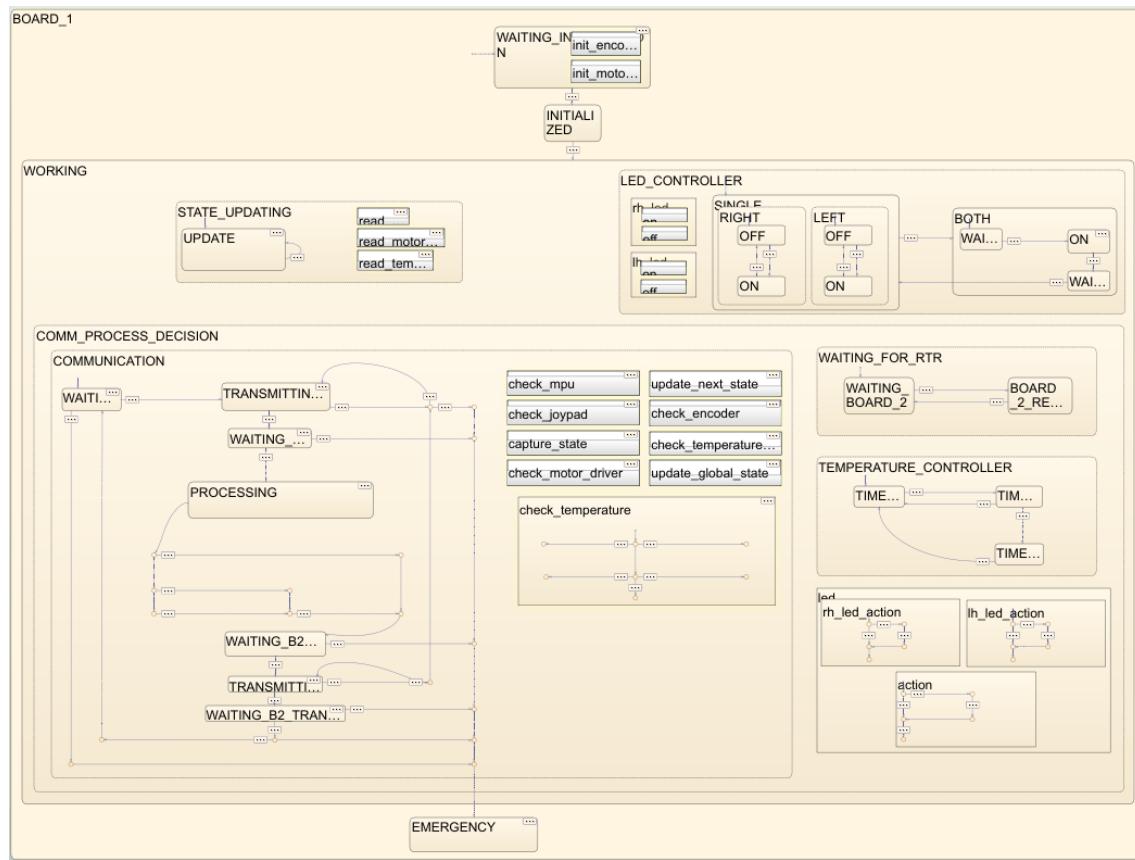


Figura 6.2: BOARD_1

Name	Type	Value	Description
RH_LED_MASK	uint8	1	Maschera per il LED destro del Rover
LH_LED_MASK	uint8	2	Maschera per il LED sinistro del Rover
BOTH_LED_MASK	uint8	4	Maschera per entrambi i LED del Rover

current_state	ROVER_STATE	Stato corrente del Rover. Valori possibili:
next_state	ROVER_STATE	Nuovo stato calcolato del Rover a valle della comunicazione tra le due board
has_retransmitted	boolean	Serve per comprendere se è avvenuta una ritrasmissione a causa di un mancato ACK da parte dell'altra board
braking	BRAKING_TYPE	Tipologia di frenata. Valori possibili:
rpm_motors	[double]	Ultimo aggiornamento sulla velocità dei motori in RPM

motors_direction	[DIRECTION]		<p>Array contenente la direzione di rotazione dei motori. Ogni elemento dell'array può assumere tre possibili valori:</p> <ul style="list-style-type: none"> • <i>DEFAULT_DIRECTION</i>: motore non in movimento • <i>BACKWARD_DIRECTION</i>: direzione anti-oraria • <i>FORWARD_DIRECTION</i>: direzione oraria
obstacle_distances	[double]		Array contenente le distanze dagli ostacoli in metri
temperature_B1	double		Temperatura della <i>BOARD_1</i> , in gradi centigradi
temperature_B2	double		Temperatura della <i>BOARD_2</i> , in gradi centigradi
accX	double		Ultimo aggiornamento sull'accelerazione lungo l'asse <i>x</i> , in m/s^2
gyroZ	double		Ultimo aggiornamento sulla velocità angolare attorno all'asse <i>z</i> , in rad/s
y_analog	int8		Valore dell'analogico del joypad lungo l'asse <i>y</i>
x_analog	int8		Valore dell'analogico del joypad lungo l'asse <i>x</i>
control_buttons	uint8		Stato dei pulsanti del joypad
control_value	uint8		Variabile utilizzata per verificare il corretto funzionamento del joypad e dell'accelerometro/giroscopio

encoder_status	ENCODER_STATUS		<p>Stato degli encoder dei motori. Valori possibili:</p> <ul style="list-style-type: none"> • <i>ENCODER_OK</i>: tutti gli encoder funzionano correttamente • <i>ENCODER_ERR</i>: errore generico degli encoder • <i>ENCODER1_ERR</i>: errore nell'encoder 1 • <i>ENCODER2_ERR</i>: errore nell'encoder 2 • <i>ENCODER3_ERR</i>: errore nell'encoder 3 • <i>ENCODER4_ERR</i>: errore nell'encoder 4
temperature_status	TEMPERATURE_STATUS		<p>Stato della temperatura. Valori possibili:</p> <ul style="list-style-type: none"> • <i>NORMAL</i>: temperatura entro i valori normali • <i>ABOVE_NORMAL</i>: temperatura al di sopra della norma • <i>CRITICAL</i>: temperatura critica
mpu_status	MPU_STATUS		<p>Stato dell'MPU6050. Valori possibili:</p> <ul style="list-style-type: none"> • <i>MPU_OK</i> • <i>MPU_ERR</i>
joypad_status	JOYPAD_STATUS		<p>Stato del joypad. Valori possibili:</p> <ul style="list-style-type: none"> • <i>JOYPAD_OK</i>: joypad connesso e funzionante • <i>JOYPAD_ERR</i>: joypad disconnesso o con errori

Macchina a stati

motor_driver_status	MOTOR_DRI-VER_STATUS		<p>Stato dei motor driver. Valori possibili:</p> <ul style="list-style-type: none"> • <i>MOTOR_DRI-VER_OK</i>: driver funzionanti • <i>MOTOR_DRI-VER_ERR</i>: errore nei driver
degraded	DEGRADED_STA-TUS		<p>Indica se il sistema è in uno stato degradato. Valori possibili:</p> <ul style="list-style-type: none"> • <i>ACTIVE</i> • <i>INACTIVE</i>
obstacle	OBSTACLE_DIR		<p>Direzione dell'ostacolo "improvviso" rilevato. Valori possibili:</p> <ul style="list-style-type: none"> • <i>NOTONE</i>: ostacolo assente in ogni direzione • <i>LEFT_SIDE</i>: ostacolo a sinistra • <i>RIGHT_SIDE</i>: ostacolo a destra • <i>FRONT</i>: ostacolo frontale • <i>LEFT_RIGHT</i>: ostacoli sia a destra che a sinistra
RH_LED_ON	Event		Evento che indica l'accensione del LED destro
RH_LED_OFF	Event		Evento che indica lo spegnimento del LED destro
LH_LED_ON	Event		Evento che indica l'accensione del LED sinistro
LH_LED_OFF	Event		Evento che indica lo spegnimento del LED sinistro
LEDS_BTN_PRESSED	Event		Evento che indica la pressione del pulsante dei LED
LEDS_BTN_RELEASED	Event		Evento che indica il rilascio del pulsante dei LED

Tabella 6.2: Variabili dello StateFlow Chart *BOARD_1*

Esamineremo ora nel dettaglio ciascuno dei suoi componenti.

6.2.1 Inizializzazione

A seguito dell'attivazione della board, vi è una fase di setup dei sensori/attuatori ad essa connessi, rappresentata dallo stato *WAITING_INITIALIZATION*, terminata la quale si entra nello stato *INITIALIZED* che funge anche da stato di sincronizzazione, dal momento in cui si attende che anche l'altra board si sia inizializzata. In particolare, per la board in esame, si ha la necessità di effettuare il setup dei soli encoder (uno per ogni motore) e dei motor driver (uno per ogni coppia di motori). Ciò viene fatto richiamando le funzioni stub `init_encoder` e `init_motor_driver` che saranno in seguito sostituite con quelle messe a disposizione dai rispettivi driver.

Nota

L'`after` presente sulla transizione che dallo stato *WAITING_INITIALIZATION* allo stato *INITIALIZED* è stato utilizzato per simulare un tempo di inizializzazione, eventualmente diverso, delle due board.

Come risultato dell'evento di sincronizzazione `[in(BOARD_2.INITIALIZED) || in(BOARD_2.WORKING)]` si entrerà nel superstato *WORKING*, descritto nella sezione successiva.

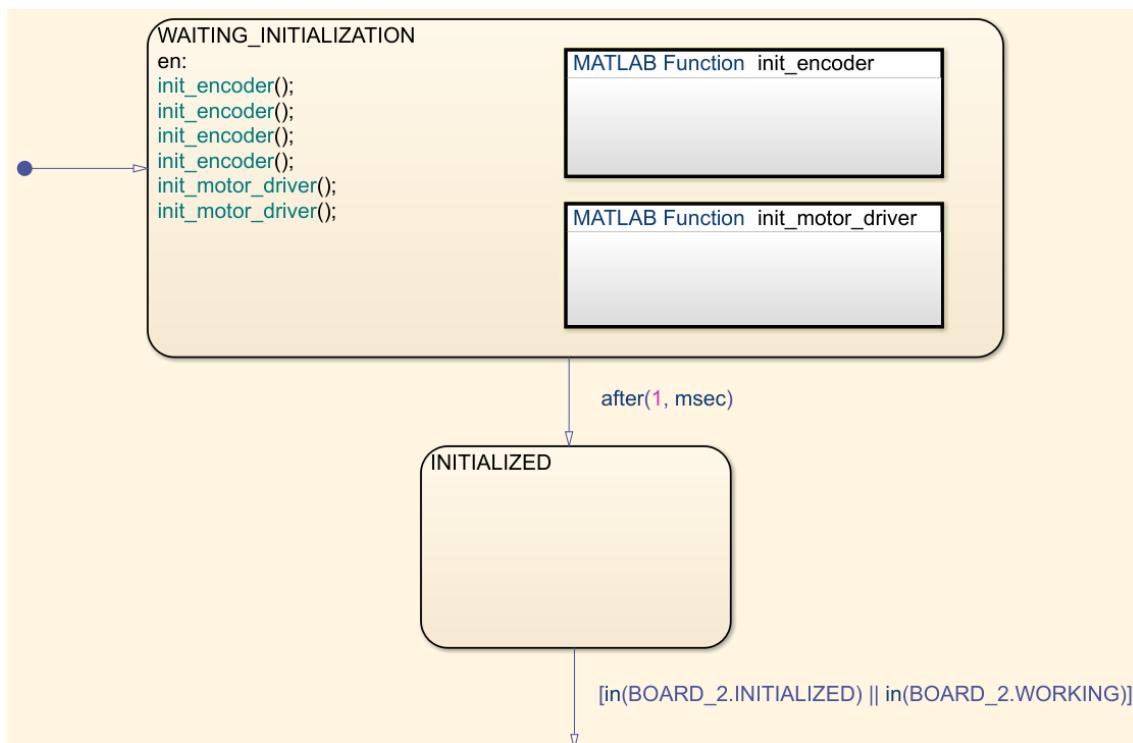


Figura 6.3: Initialization

6.2.2 Esecuzione

Il superstato *WORKING* implementa quella che è la logica principale del sistema. In particolare, è utilizzato per realizzare la decomposizione parallela

tra gli stati *STATE_UPDATING*, *COMM_PROCESS_DECISION* e *LED_CONTROLLER*, che riflettono quelli che saranno poi i task da implementare per il funzionamento del Rover.

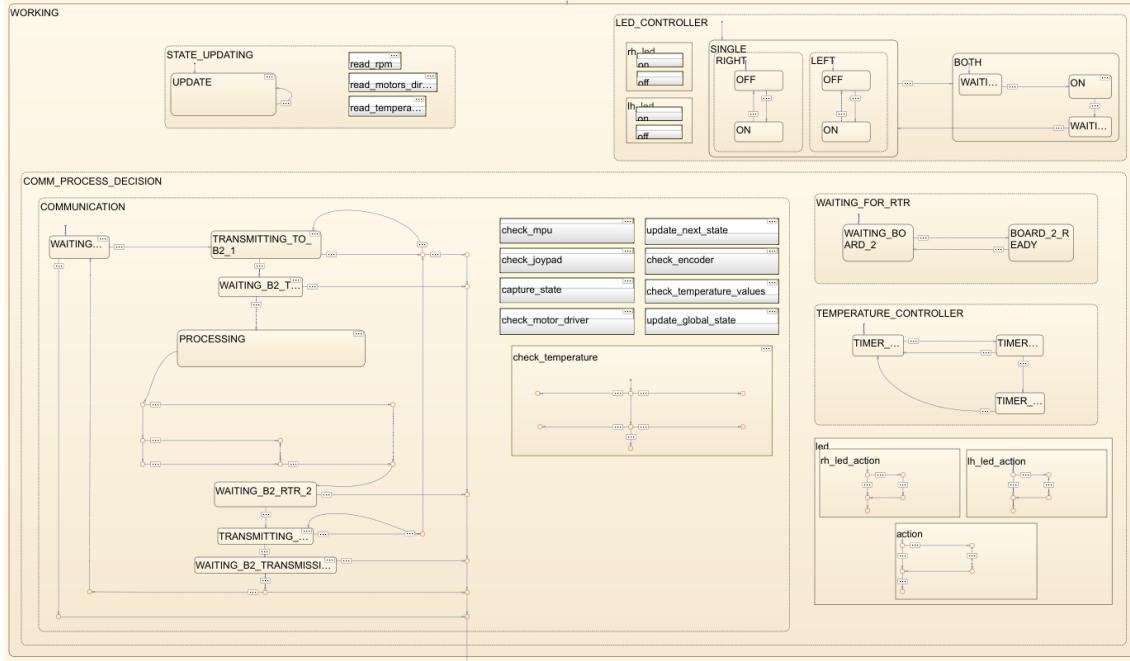


Figura 6.4: Working

STATE_UPDATING

STATE_UPDATING è costituito da uno stato *UPDATE* che viene eseguito ogni *READ_PERIOD* millisecondi ottenendo i nuovi dati dai sensori e aggiornando le rispettive variabili. In particolare, la *Board_1* si occupa di leggere i dati provenienti dagli encoder, ovvero

- gli rpm, attraverso la funzione stub `read_rpm`
- la direzione di rotazione dei motori, attraverso la funzione stub `read_motors_direction`

e la temperatura proveniente dal sensore integrato, attraverso la funzione stub `read_temperature`.

Macchina a stati

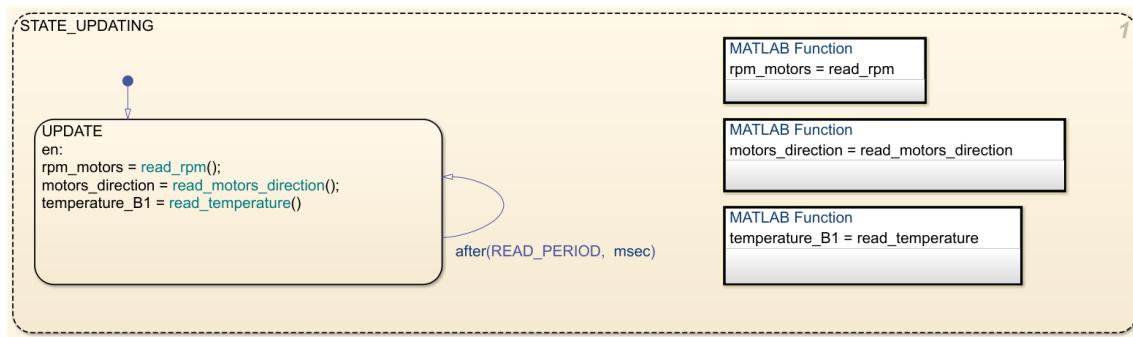


Figura 6.5: State updating

COMM_PROCESS_DECISION

COMM_PROCESS_DECISION è anch'esso in decomposizione parallela ed è costituito da tre superstati: *COMMUNICATION*, *WAITING_FOR_RTR* e *TEMPERATURE_CONTROLLER*.

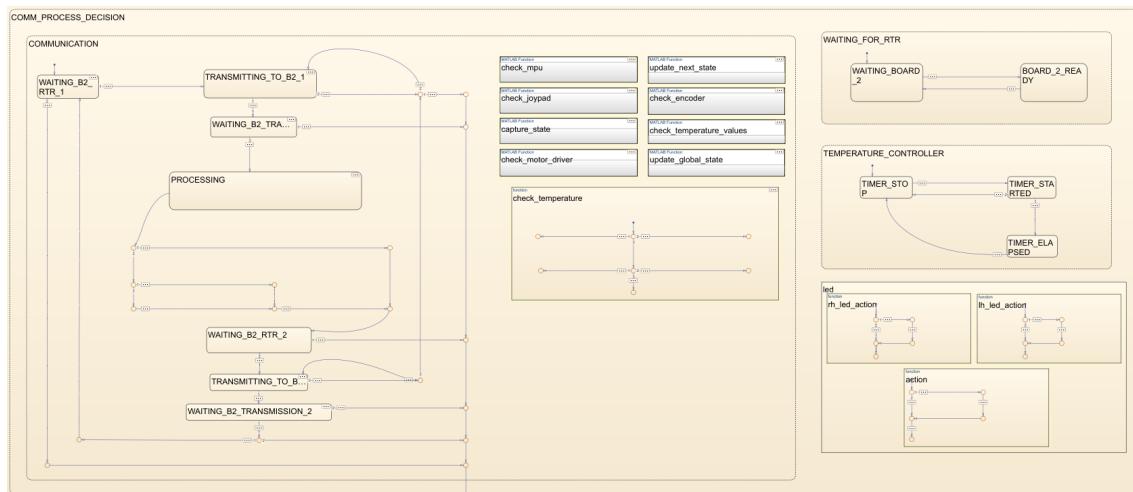


Figura 6.6: Communication and decision

Inizialmente, nel superstato *COMMUNICATION*, riportato in Figura 6.7, ci si mette in attesa che la *Board_2* inizi la comunicazione (*WAITING_B2_RTR_1*).

Macchina a stati

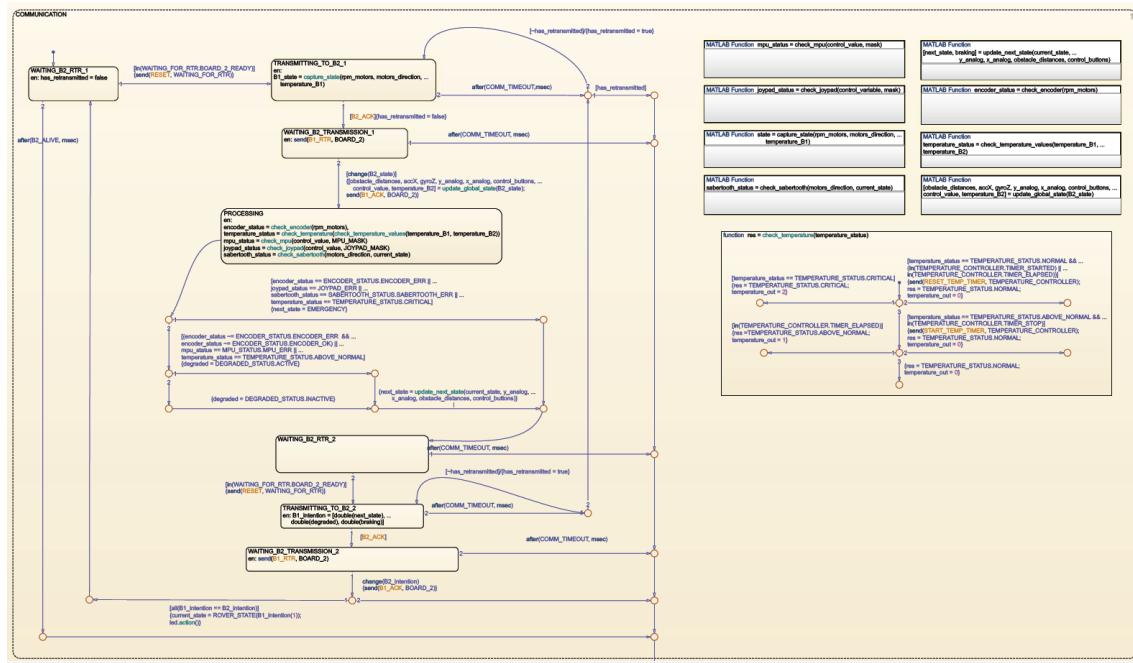


Figura 6.7: Communication

Tale evento viene catturato da un'altra macchina a stati realizzata nel superstato **WAITING_FOR_RTR**, riportata in Figura 6.8.

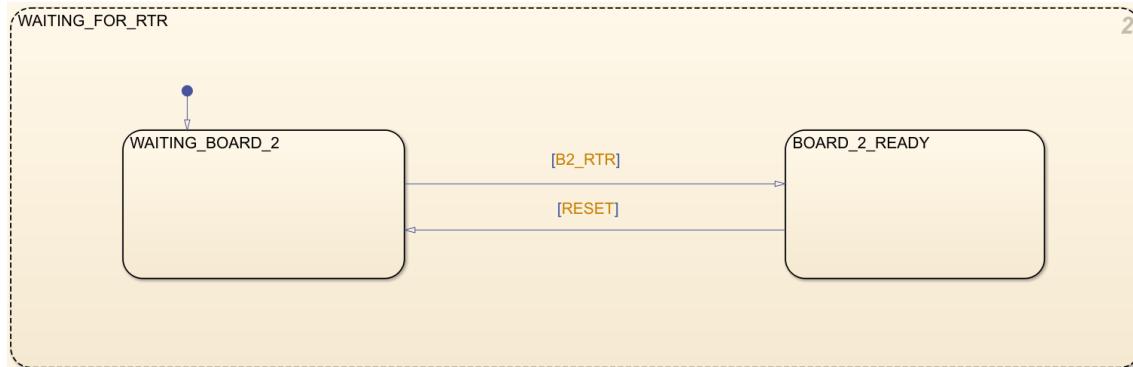


Figura 6.8: Waiting BOARD_2

Inizialmente, la macchina si troverà nello stato **WAITING_BOARD_2** e quando riceverà l'evento **B2_RTR** (**B2 Ready To Receive**) dalla **Board_2** entrerà nello stato **BOARD_2_READY**.

Quando ciò si verifica, la condizione **[in(WAITING_FOR_RTR.BOARD_2_READY)]** sarà verificata, e si passerà nello stato **TRANSMITTING_TO_B2_1** a seguito dell'azione **send(RESET, WAITING_FOR_RTR)** per indicare che la richiesta di trasmissione alla **Board_2** è stata presa in carico, riportando in uno stato di attesa la macchina **WAITING_FOR_RTR**. Nel caso in cui non arrivi l'RTR da parte della **Board_2** entro **B2_ALIVE** millisecondi, si considererà quest'ultima in failure entrando nello stato di emergenza, descritto in seguito.

A questo punto, la macchina *COMMUNICATION* avanza con la cattura dello stato attuale attraverso la funzione stub `capture_state(rpm_motors, motors_direction, temperature_B1)` assegnando quanto restituito alla variabile globale `B1_state`, che sarà poi letta dalla *Board_2*, al fine di simulare la comunicazione tra le due. Ci si mette quindi in attesa dell'ACK dalla *Board_2* ([B2_ACK]) e:

- se questo non arriva entro `COMM_TIMEOUT` millisecondi si effettua una ritrasmissione (ciò può essere dovuto ad esempio ad un errore di trasmissione) e se neanche questa va a buon fine si considera la *Board_2* in failure entrando nello stato di emergenza
- altrimenti, si entra nello stato *WAITING_B2_TRANSMISSION_1* inviando alla *Board_2* l'evento `B1_RTR`, al fine di ricevere il suo stato

Quando ciò accade ([`change(B2_state)`]), la *Board_1* aggiorna il proprio stato globale con le nuove informazioni attraverso la funzione stub `[obstacle_distances, accX, gyroZ, y_analog, x_analog, control_buttons, control_value, temperature_B2] = update_global_state(B2_state)` e invia un'ACK alla *Board_2* per confermare la ricezione (`send(B1_ACK, BOARD_2)`).

Si entra quindi nello stato *PROCESSING* in cui si effettuano i controlli necessari al fine di implementare i comportamenti richiesti dalle specifiche. Ciò viene fatto richiamando le funzioni stub

- `encoder_status = check_encoder(rpm_motors)`
- `temperature_status = check_temperature(check_temperature_values(temperature_B1, temperature_B2))`
- `mpu_status = check_mpu(control_value, MPU_MASK)`
- `joypad_status = check_joypad(control_value, JOYPAD_MASK)`
- `motor_driver_status = check_motor_driver(motors_direction, current_state)`

Con particolare riferimento alla gestione della temperatura, il comportamento implementato è il seguente:

- se è compresa tra `MIN_NORMAL_TEMP` e `MAX_NORMAL_TEMP` il sistema può funzionare normalmente
- se è maggiore di `MAX_NORMAL_TEMP` e minore di `CRTICAL_TEMP` per `TEMP_TIMEOUT` secondi, il sistema deve funzionare in maniera degradata
- se è maggiore di `CRTICAL_TEMP`, il sistema deve andare nello stato di emergenza

Nota

Come temperatura utilizzata per effettuare il check si utilizza quella massima tra quelle delle due board. Inoltre, al fine di verificare il corretto funzionamento dei sensori delle due schede, viene controllato anche che i due valori di temperatura differiscano al massimo di un certo valore.

La macchina a stati utilizzata per monitorare il tempo in cui la temperatura è nell'intervallo $[\text{MAX_NORMAL_TEMP}, \text{CRITICAL_TEMP}]$ è implementata nel superstato *TEMPERATURE_CONTROLLER* riportato in Figura 6.9.

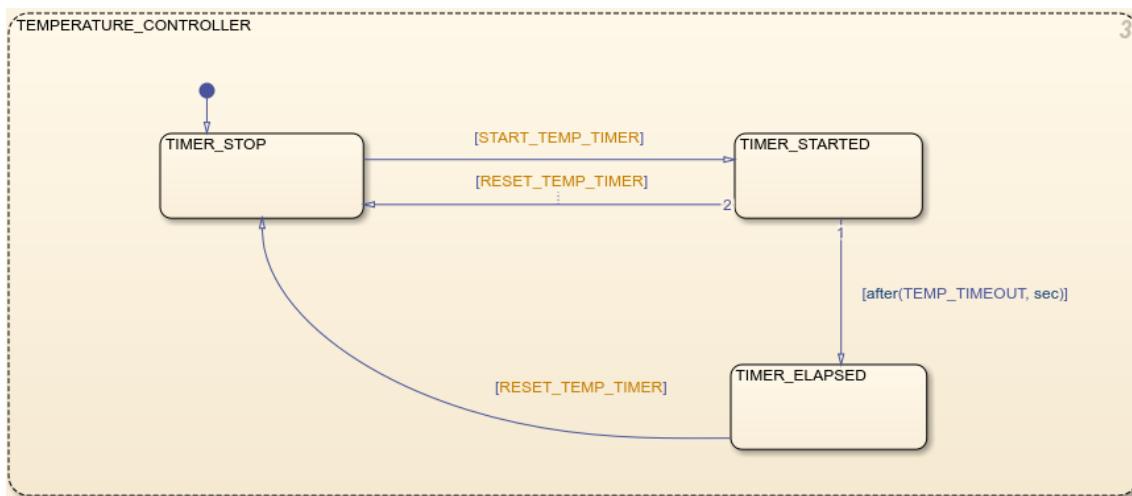


Figura 6.9: Temperature controller

Inizialmente, la macchina si trova nello stato *TIMER_STOP*. Non appena la temperatura raggiunge l'intervallo precedentemente definito, la funzione *check_temperature* effettua la send dell'evento *START_TEMP_TIMER*, avviando il conteggio del tempo transitando nello stato *TIMER_STARTED*. Nell'ipotesi in cui la temperatura ritorni normale prima dello scadere del timer, questo viene fermato inviando l'evento *RESET_TEMP_TIMER*, altrimenti, al timeout (ovvero all'attivazione della transizione associata all'evento *[after(TEMP_TIMEOUT, sec)]*) la macchina transiterà nello stato *TIMER_ELAPSED*, segnalando alla funzione *check_temperature* la necessità di portare il sistema in uno stato di funzionamento degradato.

Terminati i suddetti controlli, sulla base delle condizioni esposte nei paragrafi 2.2 e 2.3 del Capitolo 2, si deciderà il prossimo stato, calcolato dalla funzione *stub next_state = update_next_state(current_state, y_analog, x_analog, obstacle_distances, control_buttons)* se questo è diverso da quello di emergenza, e si settterà la variabile *degraded* in base all'esigenza di funzionamento in modalità degradata.

Successivamente, seguendo lo stesso pattern utilizzato per lo scambio delle informazioni di stato, le due board si comunicano la loro intenzione, ovvero, come vogliono che il sistema evolva e, per quanto concerne la board in esame,

si verifica che le due intenzioni coincidano ([all(B1_intention == B2_intention)]), aggiornando di conseguenza lo stato corrente del Rover, inviando uno specifico evento alla macchina che si occupa del controllo dei led (current_state = ROVER_STATE(B1_intention(1)); led.action()) e ritornando nello stato WAITING_B2_RTR_1 attendendo l'inizio di una nuova comunicazione. In caso negativo, il sistema si porta nello stato di emergenza.

Per una migliore comprensione di quanto descritto, si riportano le implementazioni delle funzioni grafiche `check_temperature` e quelle inerenti l'invio degli eventi per il controllo dei led.

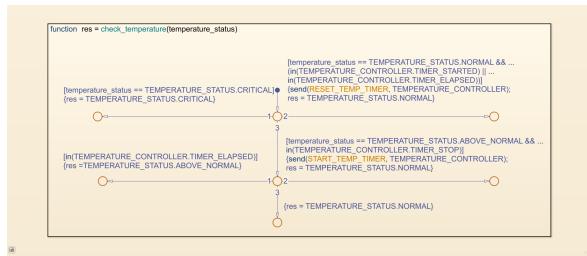


Figura 6.10: `check_temperature` function

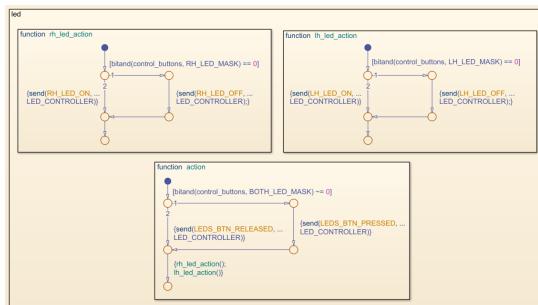


Figura 6.11: led control functions

Nota

Al fine di implementare una sorta di meccanismo di heartbeat per assicurarsi che le due board stiano funzionando, senza l'introduzione di ulteriori collegamenti tra le due, da ogni stato del superstato `COMMUNICATION` diverso da quello di trasmissione, si attende un tempo pari a `COMM_TIMEOUT` millisecondi. Se entro questo tempo ciascuna delle due board non esegue l'operazione attesa dall'altra (nel caso in esame, la `Board_2`, si considera quest'ultima in failure andando nello stato di emergenza).

Nota

La modellazione attuale del protocollo di scambio delle informazioni prevede la possibilità di effettuare al massimo una ritrasmissione per ciascuna comunicazione. Tuttavia, al fine di migliorare la durata complessiva nel worst-case del protocollo, come mostrato nel paragrafo 4.1.2 del Capitolo 4, è possibile modificare semplicemente la macchina che modella tale comportamento (*COMMUNICATION*) permettendo al massimo una ritrasmissione in totale.

LED_CONTROLLER

Il superstato *LED_CONTROLLER*, rappresentato in Figura 6.12, vuole modellare quello che sarà poi il task di attuazione della *Board_7* che ha semplicemente il compito di controllare l'accensione e lo spegnimento dei led sulla base dei tasti premuti sul joypad e di quanto accordato dalle due board in fase di comunicazione.

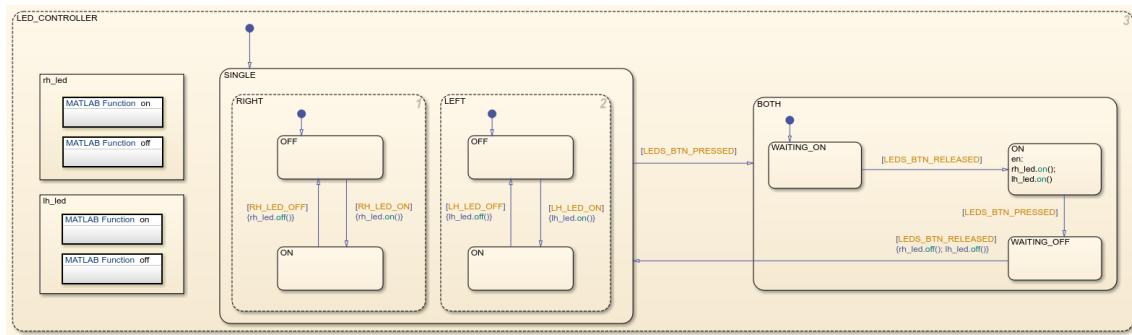


Figura 6.12: Led controller

La macchina presenta al suo interno due stati principali:

- *SINGLE*: ha il compito di gestire l'accensione e lo spegnimento dei singoli led. È in decomposizione parallela per governare simultaneamente il led destro (superstato *RIGHT*) e quello sinistro (superstato *LEFT*)
- *BOTH*: ha il compito di gestire l'accensione e lo spegnimento di entrambi i led. In particolare, sulla pressione del bottone relativo all'accensione di entrambi i led si passa dallo stato *SINGLE* allo stato *BOTH* attendendo il suo rilascio (*WAITING_ON*), momento in cui si entra nello stato *ON* con conseguente attivazione dei led. Nel momento in tale bottone viene premuto nuovamente, si entra nello stato *WAITING_OFF* in cui si attende il suo rilascio, momento in cui vengono spenti i led ritornando nel superstato *SINGLE*.

Per gestire l'accensione e lo spegnimento dei led sono state definite le funzioni stub *rh_led.on()* e *rh_led.off()* per quello destro e *lh_led.on()* e *lh_led.off()* per quello sinistro.

6.3 BOARD_2

Passiamo adesso alla descrizione della *Board_2* che, a differenza della *Board_1*, svolge il ruolo di una sorta di "master", dal momento in cui, oltre a ottenere i dati dai sensori ad essa connessi e aggiornare le proprie variabili di stato, è lei a decidere quando iniziare la comunicazione con la *Board_1* al fine di stabilire di comune accordo il prossimo stato fisico del Rover e procedere con l'attuazione. Una panoramica dell'intera *Board_2* è mostrata nella Figura 6.13.

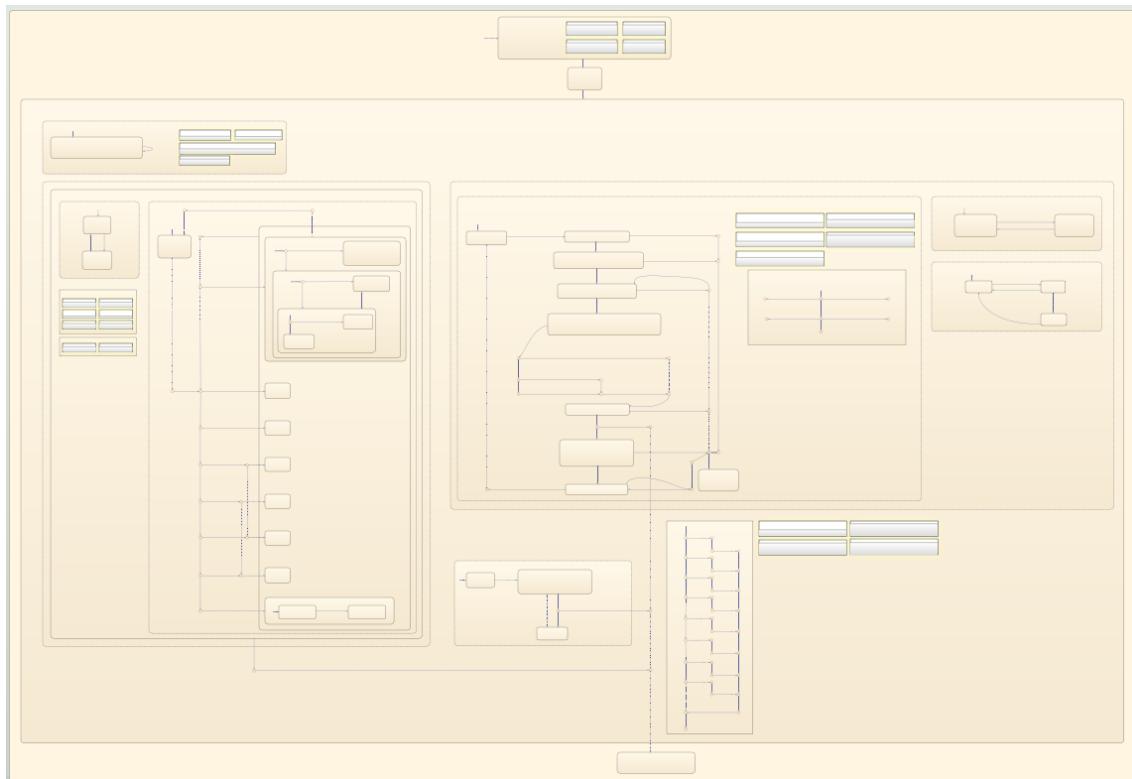


Figura 6.13: BOARD_2

Name	Type	Value	Description
EMERGENCY_STOP_MASK	uint8	32	Maschera associata al pulsante per lo stop di emergenza
current_state	ROVER_STATE		Come BOARD_1
next_state	ROVER_STATE		Come BOARD_1
has_retransmitted	boolean		Come BOARD_1
braking	BRAKING_TYPE		Come BOARD_1
rpm_motors	[double]		Come BOARD_1
motors_direction	[DIRECTION]		Come BOARD_1
obstacle_distances	[double]		Come BOARD_1

Macchina a stati

temperature_B1	double		Come BOARD_1
temperature_B2	double		Come BOARD_1
accX	double		Come BOARD_1
gyroZ	double		Come BOARD_1
y_analog	int8		Come BOARD_1
x_analog	int8		Come BOARD_1
control_buttons	uint8		Come BOARD_1
control_value	uint8		Come BOARD_1
encoder_status	ENCODER_STATUS		Come BOARD_1
temperature_status	TEMPERATURE_STATUS		Come BOARD_1
mpu_status	MPU_STATUS		Come BOARD_1
joypad_status	JOYPAD_STATUS		Come BOARD_1
motor_driver_status	MOTOR_DRIVER_STATUS		Come BOARD_1
degraded	DEGRADED_STATUS		Come BOARD_1
obstacle	OBSTACLE_DIR		Come BOARD_1
IDLE	Event		Evento che indica lo stop del Rover
FORWARD	Event		Evento che indica l'avanzamento del Rover in avanti
FORWARD_RIGHT	Event		Evento che indica l'avanzamento del Rover in avanti-destra
FORWARD_LEFT	Event		Evento che indica l'avanzamento del Rover in avanti-sinistra
RIGHT	Event		Evento che indica la rotazione del Rover sul bari-centro verso destra
LEFT	Event		Evento che indica la rotazione del Rover sul bari-centro verso sinistra
BACKWARD	Event		Evento che indica l'avanzamento del Rover all'indietro
ALT_BACKWARD	Event		Evento che indica l'attuazione della retromarcia alternativa

ACTIVE	Event		Evento che indica l'attivazione della modalità "single board" quando la <i>BOARD_1</i> va in failure
--------	-------	--	--

Tabella 6.3: Variabili dello StateFlow Chart *BOARD_1*

Come fatto per la *Board_1*, esamineremo ora nel dettaglio ciascuno dei suoi componenti.

6.3.1 Inizializzazione

Quanto detto nel paragrafo 6.2.1 per la *Board_1* vale analogamente per la *Board_2*. Ciò in cui differisce è che in questo caso si ha la necessità di effettuare il setup dei motor driver, del joypad, degli ultrasuoni e dell'accelerometro/giroscopio. Ciò viene fatto mediante chiamate alle funzioni stub `init_motor_driver`, `init_ps2x`, `init_ultrasonic_sensors` e `init_mpu` che saranno in seguito sostituite con quelle messe a disposizione dai rispettivi driver.

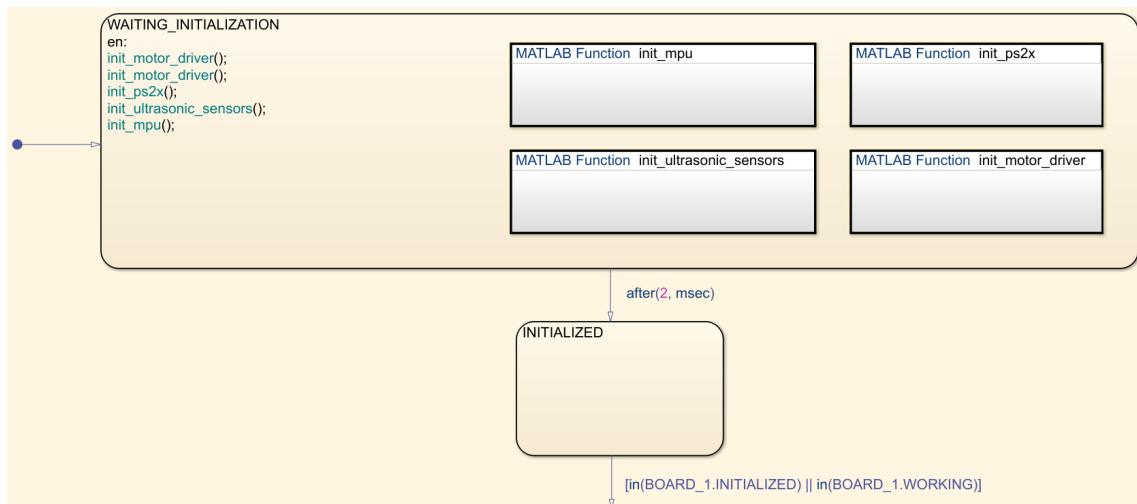


Figura 6.14: Initialization

6.3.2 Esecuzione

Anche in questo caso, il superstato *WORKING* implementa quella che è la logica principale del sistema. In particolare, è utilizzato per realizzare la decomposizione parallela tra gli stati *STATE_UPDATING*, *COMM_PROCESS_DECISION*, *EXECUTION* e *SINGLE_BOARD*, che riflettono quelli che saranno poi i task da implementare per il funzionamento del Rover.

Macchina a stati

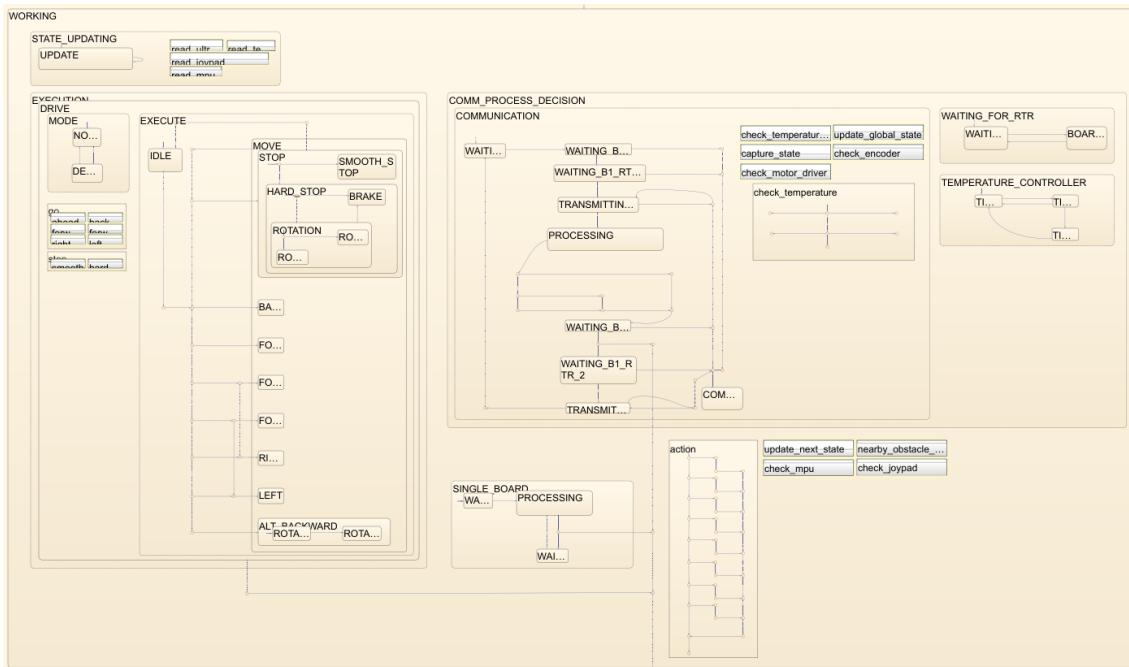


Figura 6.15: Working

STATE_UPDATING

Anche in questa circostanza si applica quanto già detto per la *Board_7*. In particolare, la *Board_2* si occupa di leggere:

- le distanze dagli ostacoli, attraverso la funzione stub `read_ultrasonic_sensors`
- l'accelerazione lineare e angolare, attraverso la funzione stub `read_mpu`
- il joypad, attraverso la funzione stub `read_joystick`
- la temperatura proveniente dal sensore in essa integrato, attraverso la funzione stub `read_temperature`

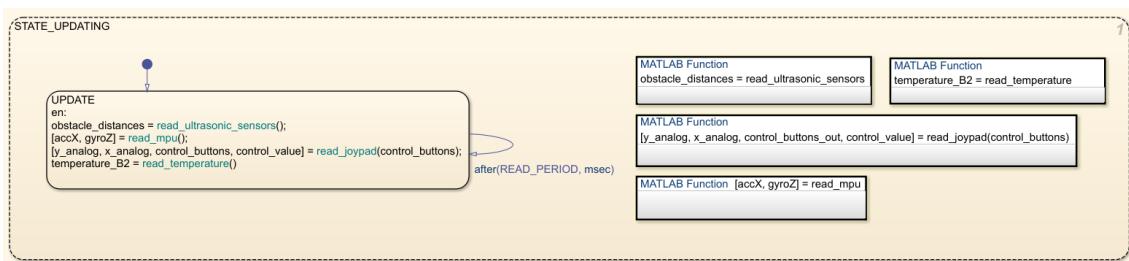


Figura 6.16: State updating

COMM_PROCESS_DECISION

Il superstato *COMM_PROCESS_DECISION* della *Board_2*, analogamente a quello della *Board_1*, è in decomposizione parallela ed è anch'esso composto dai tre superstati *COMMUNICATION*, *WAITING_FOR_RTR* e *TEMPERATURE_CONTROLLER*, così come mostrato in Figura 6.17.

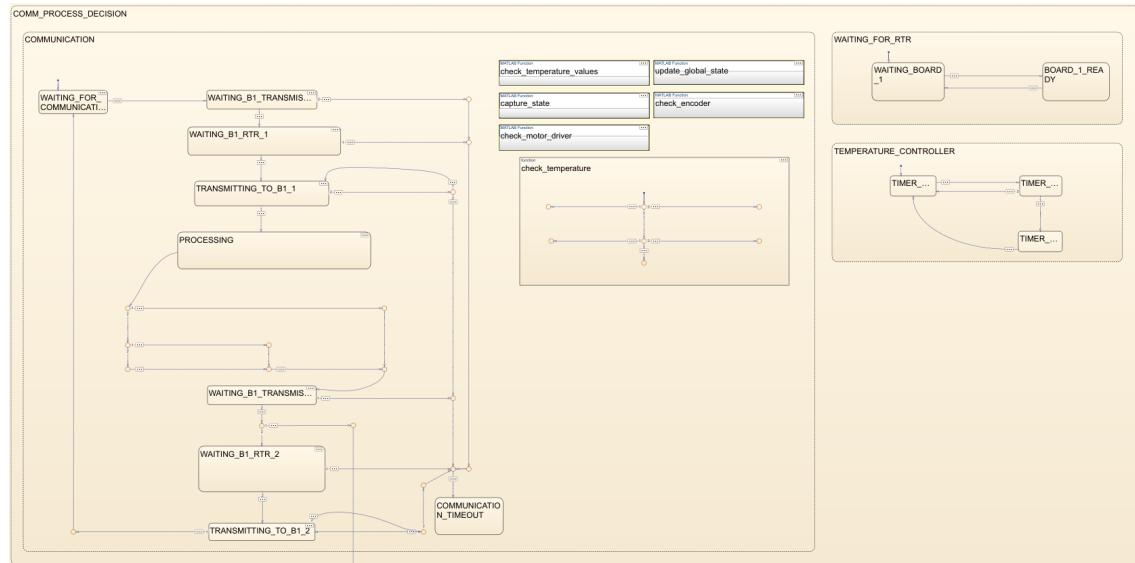


Figura 6.17: Communication and decision

Mentre questi ultimi due sono identici a quelli già presentati per la *BOARD_1*, il superstato *COMMUNICATION* presenta alcune differenze, essendo il complementare dell'altro.

Macchina a stati

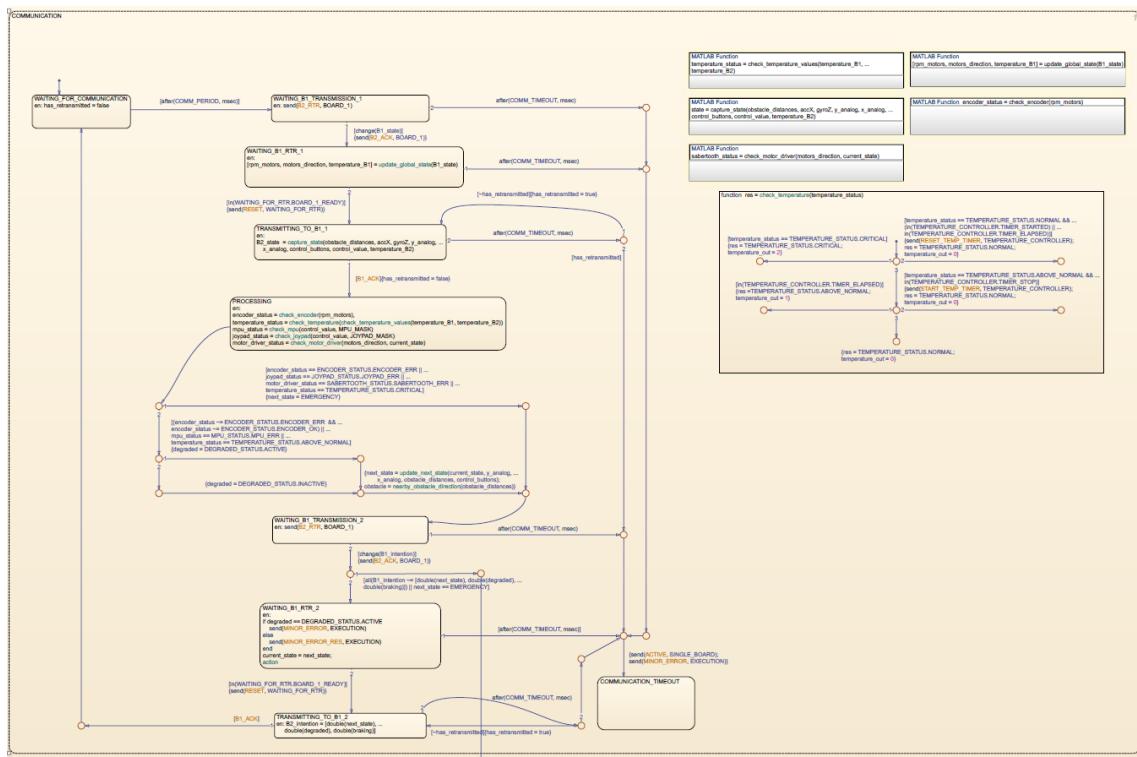


Figura 6.18: Communication

In particolare, dal momento in cui la *Board_2* opera da "master", inizialmente si trova nello stato *WAITING_FOR_COMMUNICATION* e inizia la comunicazione ogni *COMM_PERIOD* millisecondi.

Una seconda differenza la troviamo poi nell'insieme di eventi che vengono inviati a seguito della comunicazione e accordo del prossimo stato fisico tra le due board. Infatti, mentre la *Board_1* ha la sola responsabilità di pilotare l'accensione dei led, è la *Board_2* a gestire il movimento del Rover e il suo funzionamento in modalità degradata. Pertanto, se a seguito della comunicazione le intenzioni delle due board sono diverse o se è stato deciso di comune accordo che il prossimo stato è quello di emergenza, si entra in tale stato, altrimenti si invia l'evento `MINOR_ERROR` o `MINOR_ERROR_RES` a seconda del caso in cui si deve entrare in uno stato di funzionamento degradato o meno, e uno degli eventi che indicano lo stato fisico in cui deve portarsi il Rover. Tale operazione viene eseguita dalla graphical function `action` che, sulla base dello stato corrente concordato effettua la send dello specifico evento.

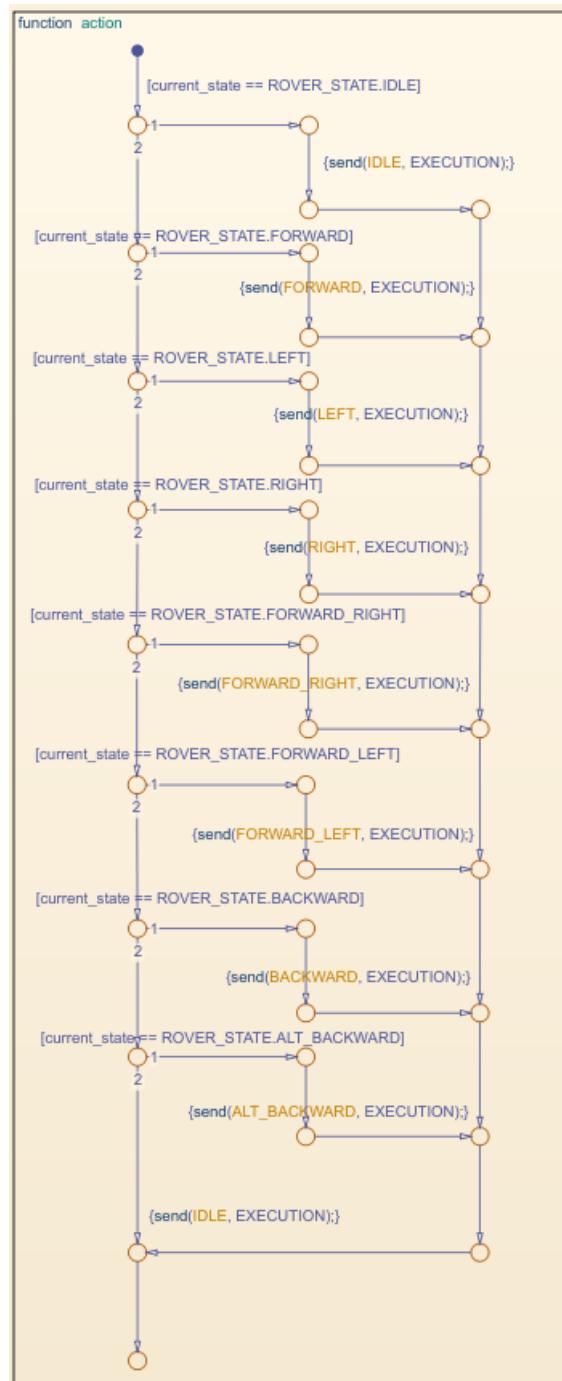


Figura 6.19: Action

Un'ultima differenza è la seguente: mentre la *Board_1* reagisce a un failure della *Board_2* semplicemente entrando in uno stato di emergenza e fermando i motori, la *Board_2* reagisce a un failure della *Board_1* entrando in modalità di funzionamento degradato, arrestando la comunicazione con essa. Ciò è modellato dallo stato *COMMUNICATION_TIMEOUT* in cui si entra a seguito dello scadere di un timer in una delle varie fasi della comunicazione,

dal momento in cui la *Board_1* si considera in failure se non esegue le operazioni attese dalla *Board_2*. Un esempio potrebbe essere il seguente: la *Board_2* è in attesa dell'RTR hardware dalla *Board_1*, ma il filo ad esso associato si è scollegato. In tale circostanza, dopo aver atteso un certo periodo di tempo, si può considerare la *Board_1* in failure, portando il sistema in uno stato di funzionamento degradato.

Pertanto, l'ingresso nello stato *COMMUNICATION_TIMEOUT* è accompagnato dalla send degli eventi *MINOR_ERROR* e *ACTIVE*. In particolare, quest'ultimo fa transitare la macchina presente nel superstato *SINGLE_BOARD*, rappresentata in Figura 6.20, dallo stato *WAITING_START* allo stato *PROCESSING* con immediato spegnimento della *Board_1*. Questo stato viene periodicamente attivato al fine di decidere, sulla base delle informazioni a disposizione della board, se portarsi in uno stato di emergenza, ad esempio a causa della perdita di connessione del joypad, oppure calcolare il prossimo stato fisico e proseguire in modalità degradata.

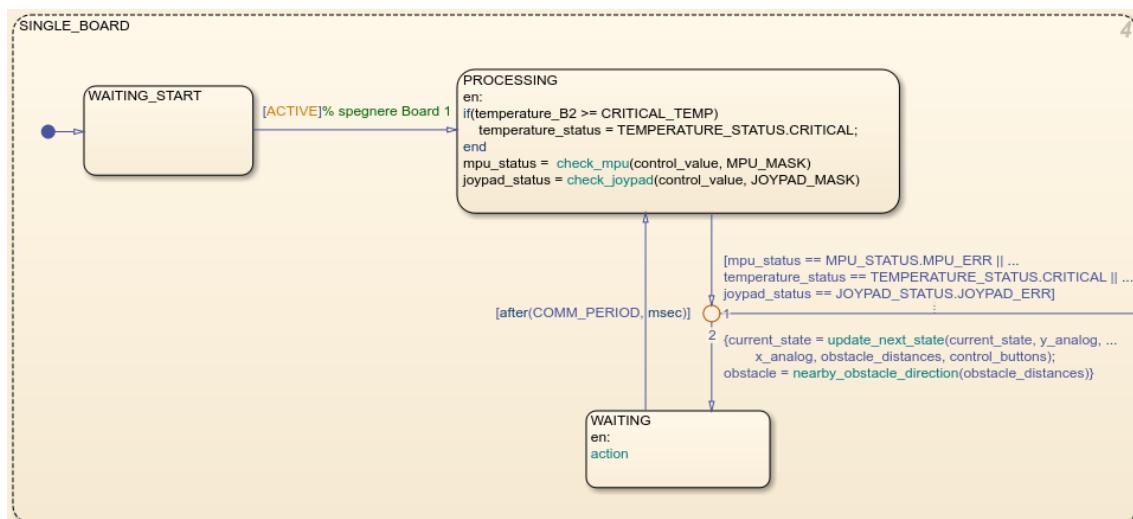


Figura 6.20: Single board

EXECUTION

All'interno di questo superstato è modellata la logica legata all'attuazione, al fine di portare il Rover nello stato fisico concordato dalle due board.

Macchina a stati

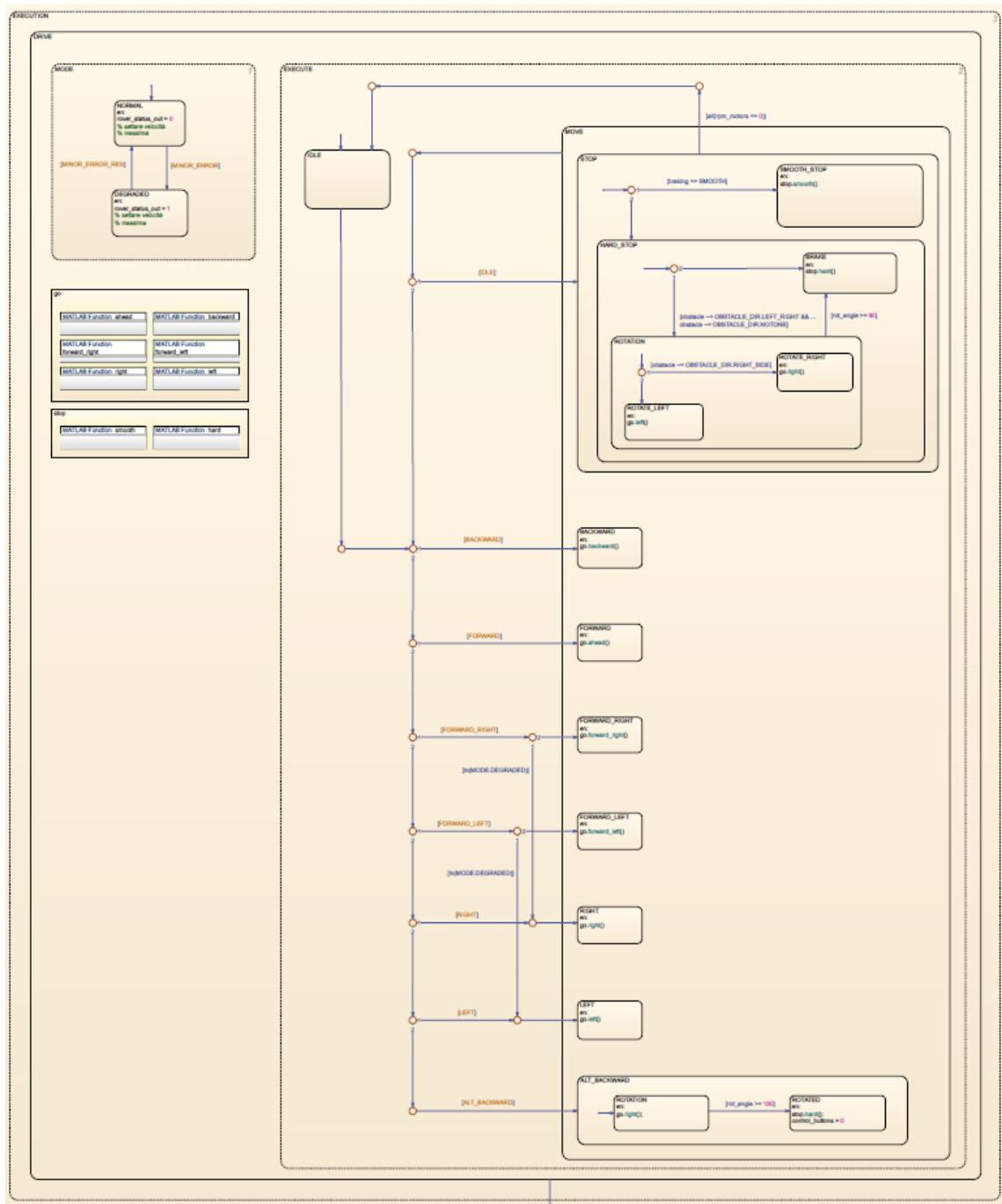


Figura 6.21: Execution

In particolare, al suo interno troviamo il superstato *DRIVE* in decomposizione parallela, contenente a sua volta i superstati *MODE* ed *EXECUTE*.

Nota

La necessità di questo superstato è legata al fatto che in qualsiasi stato fisico si trovi il Rover, se viene premuto sul joypad il tasto di arresto di emergenza, il Rover deve portarsi immediatamente in tale stato. Ciò viene fatto attraverso la super-transizione [bitand(control_buttons, EMERGENCY_STOP_MASK) ~= 0].

Il superstato *MODE* presenta due sottostati, *NORMAL* (che coincide anche con lo stato iniziale) e *DEGRADED*. Sulla base degli eventi che vengono inviati a seguito dall'accordo tra le due board, si passa dallo stato *NORMAL* a *DEGRADED* e viceversa, a seconda del caso in cui bisogna portare il Rover in modalità degradata o recuperare da essa. Ad esempio, se il sistema si trova nello stato di funzionamento degradato a causa di un aumento di temperatura, ma successivamente il suo valore non continua a salire ma rientra nell'intervallo di funzionamento ottimale, il sistema può ritornare a funzionare normalmente.

Il superstato *EXECUTE*, invece, è costituito da due stati principali, *IDLE* (che coincide con lo stato iniziale) e *MOVE*. Se da *IDLE* si riceve uno degli eventi di movimento, si transita nel rispettivo stato eseguendo l'azione che consente di portare il Rover nel suddetto stato fisico. In particolare, nel caso di retromarcia alternativa (*ALT_BACKWARD*), come da specifiche, si esegue anzitutto una rotazione di 180 gradi prima di fermarsi in attesa del comando successivo.

Grazie alla transizione in uscita dallo stato *MOVE* è possibile spostarsi in qualsiasi altro stato sulla base dell'evento ricevuto (ad esempio, all'istante attuale si sta andando avanti (*FORWARD*) e a quello successivo si vuole andare a destra *RIGHT*). Particolare attenzione merita il superstato *STOP* che si attiva a seguito della ricezione dell'evento *IDLE*. Ciò può avvenire o perché entrambi gli analogici del joypad sono stati portati nella posizione di riposo, o perché è stata rilevata la presenza di un ostacolo (sia esso "lontano" o "improvviso"). In particolare,

- se è presente un ostacolo "lontano" si effettua una frenata *smooth*
- se è presente un ostacolo "improvviso" si agisce nel seguente modo:
 - se si trova sia a destra che a sinistra, si frena bruscamente
 - se si trova a destra, si ruota di 90 gradi verso sinistra e poi si frena
 - se si trova avanti o a sinistra, si ruota di 90 gradi verso destra e poi si frena
- se non è presente alcun ostacolo, si frena bruscamente

In ogni caso, terminata la frenata, attraverso la transizione ([all(rpm_motors == 0)]), si ritorna nello stato di *IDLE*.

Anche in questo caso, le azioni di movimento e frenata sono state definite attraverso delle funzioni stub, di seguito riportate.

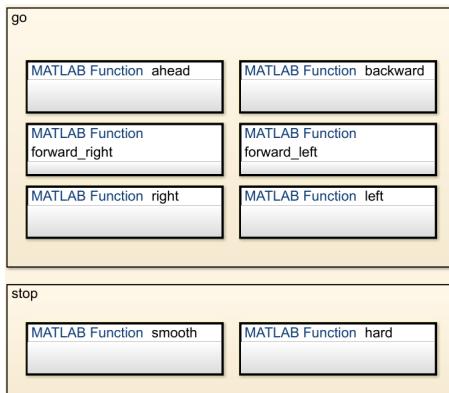


Figura 6.22: Move and stop functions

6.4 Board_1 vs Board_2: Emergenza

Come già anticipato nelle sezioni precedenti, al verificarsi di particolari condizioni (talune dettate dalle specifiche) è necessario portare il sistema in uno stato di emergenza. In particolare, per la *Board_2*, questo consiste nello spegnere la *Board_1* e arrestare i motori, al fine di portare il Rover in uno stato sicuro. La necessità di spegnere l'altra board deriva dal fatto che non si vuole che entrambe agiscano sui motor driver, ma soltanto in modalità esclusiva.

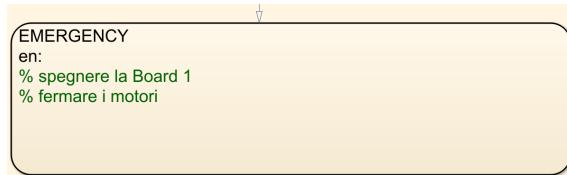


Figura 6.23: Board_2 - Emergency

Per quanto concerne la *Board_1* invece, il suo stato di emergenza consiste soltanto nell'arrestare i motori. Il motivo per cui la *Board_1* non può spegnere la *Board_2* è mostrato dal seguente esempio. Supponiamo che a valle dell'invio/ricezione delle informazioni di stato tra le due board e calcolo delle rispettive intenzioni circa il prossimo stato fisico del Rover, il collegamento RTR della *Board_2* si interrompe. Quindi, la *Board_2* è in attesa dell'intenzione della *Board_1*, mentre la *Board_1* sta aspettando l'RTR della *Board_2*. Dopo un certo intervallo di tempo, la *Board_1* assume quindi che la *Board_2* sia in failure, portandosi nel suo stato di emergenza. Quasi nello stesso momento, anche la *Board_2* considera la *Board_1* in failure (come conseguenza della mancata ricezione della sua intenzione) spegnendo la *Board_1* e portando il sistema in uno stato di funzionamento degradato. Notiamo come se la *Board_1* avesse spento la *Board_2* ciò non sarebbe stato possibile. Nel caso in cui, invece, la *Board_2* fosse stata realmente in failure, ad esempio a causa dell'interruzione della sua alimentazione, la *Board_1* avrebbe arrestato semplicemente i motori, portando il sistema in uno stato sicuro.

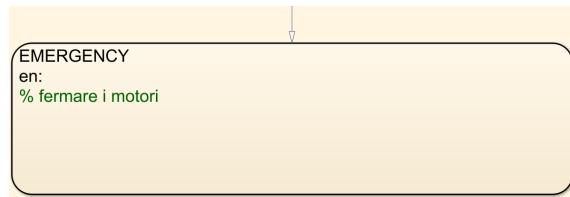


Figura 6.24: Board_1 - Emergency

6.5 Alcuni esempi di funzionamento

Al fine di simulare alcune condizioni di funzionamento del Rover, al chart Stateflow sono state aggiunte delle variabili di input e output rappresentative dei sensori/attuatori del sistema e si è fornita una grossolana implementazione di alcune funzioni stub per consentire l'evoluzione dell'intera macchina. Il sistema così ottenuto è rappresentato nella seguente figura.

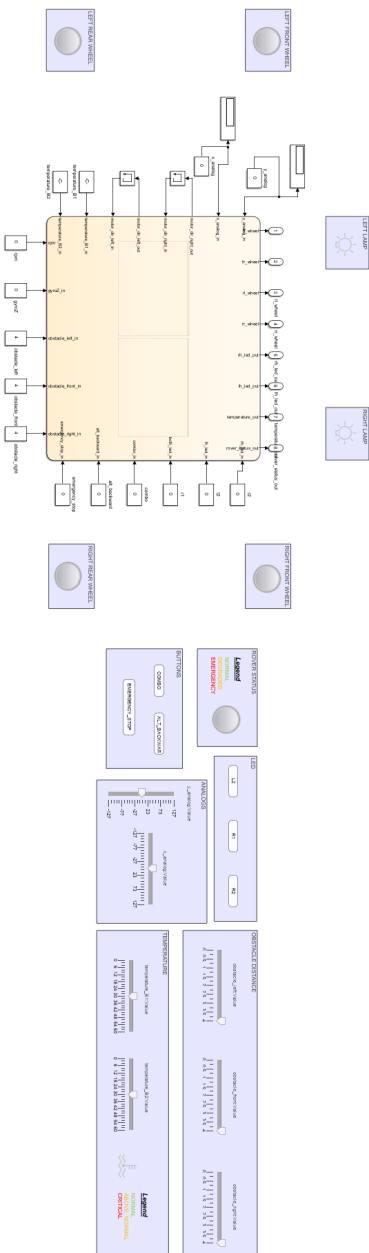


Figura 6.25: Modified chart for simulation

In particolare, il significato dei riquadri è di seguito riportato:

- ROVER STATUS: rappresenta lo stato di funzionamento del Rover; può assumere tre colori, verde, giallo, rosso, che indicano rispettivamente se il sistema sta funzionando in modalità normale, degradata o è in emergenza

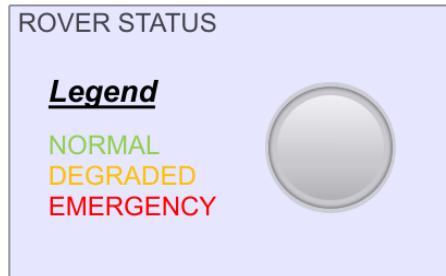


Figura 6.26: Rover status

- LED: contiene i pulsanti per l'accensione dei led; più specificamente:
 - L2 ed R2, se mantenuti premuti, consentono rispettivamente l'attivazione dei led sinistro e destro
 - R1, se premuto e rilasciato, consente l'attivazione di entrambi i led (lo stesso vale per lo spegnimento)



Figura 6.27: Led button

- BUTTONS: contiene i pulsanti per l'attivazione delle altre funzionalità; più specificamente:
 - COMBO, quando premuto, simula la pressione di un insieme di pulsanti del joypad per lo sblocco della retromarcia alternativa
 - ALT_BACKWARD, quando premuto, consente l'effettiva attivazione della retromarcia alternativa a seguito del suo sblocco
 - EMERGENCY_STOP, quando premuto, porta il sistema in uno stato sicuro

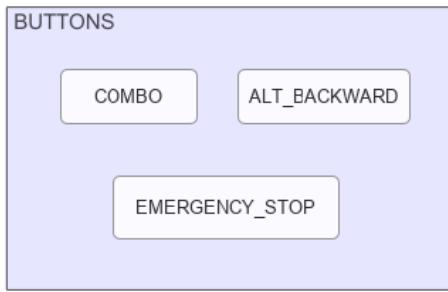


Figura 6.28: Buttons for other functionality

- **ANALOGS:** contiene due slider, uno associato alla levetta sinistra del joypad (*y_analog*) e uno a quella destra (*x_analog*), che mappano sul range [-127,127] la velocità del Rover

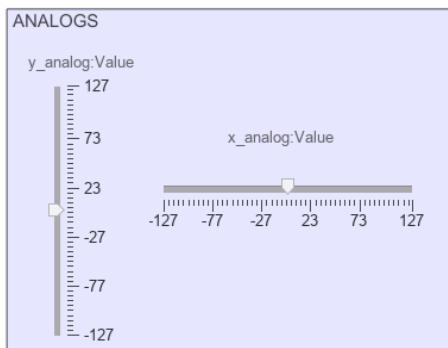


Figura 6.29: Analogs slider

- **OBSTACLE DISTANCE:** contiene tre slider, mappati sul range [0,4], che consentono di simulare la distanza (in metri) da un ostacolo in ogni direzione

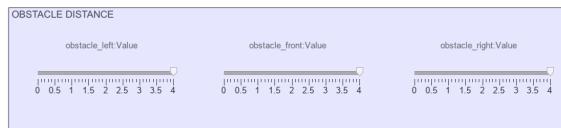


Figura 6.30: Obstacle distance slider

- **TEMPERATURE:** contiene due slider, mappati sul range [0,60], che consentono di simulare la temperatura letta dalle due board; in particolare, il colore del simbolo a destra degli slider indica se questa è normale (verde), al di sopra del normale (giallo) e critica (rosso)

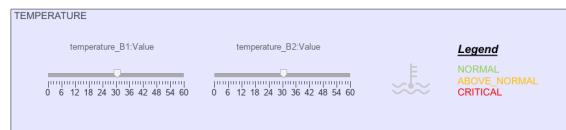


Figura 6.31: Temperature slider

- *LEFT FRONT WHEEL, RIGHT FRONT WHEEL, LEFT REAR WHEEL e RIGHT REAR WHEEL* sono stati utilizzati per simulare il verso di rotazione dei motori; più specificamente:
 - nero, indica motori fermi
 - verde, indica la rotazione dei motori in senso orario
 - rosso, indica la rotazione dei motori in senso antiorario
 - blu, è un colore speciale utilizzato per distinguere il movimento "avanti destra" da "destra" e "avanti sinistra" da "sinistra"



(a) Left front wheel



(b) Right front wheel



(c) Left rear wheel



(d) Right front wheel

Nota

È possibile settare gli rpm e l'angolo di rotazione intorno al baricentro attraverso le rispettive caselle in input al chart.

6.5.1 Scenario 1: avanzamento nella direzione avanti-destra con ostacolo improvviso a destra

Partendo dallo stato di *IDLE* (motori fermi)

Macchina a stati

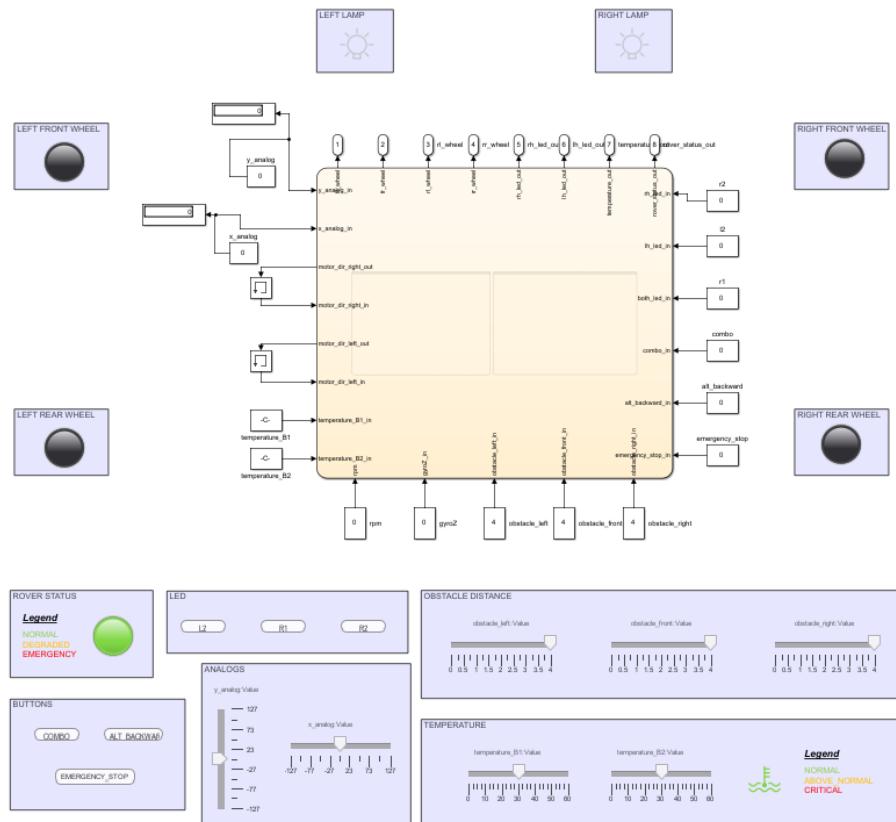


Figura 6.33: Idle

aumentiamo la velocità in entrambe le direzioni attraverso gli slider associati agli analogici, in modo da portarci nello stato **FORWARD_RIGHT**

Macchina a stati

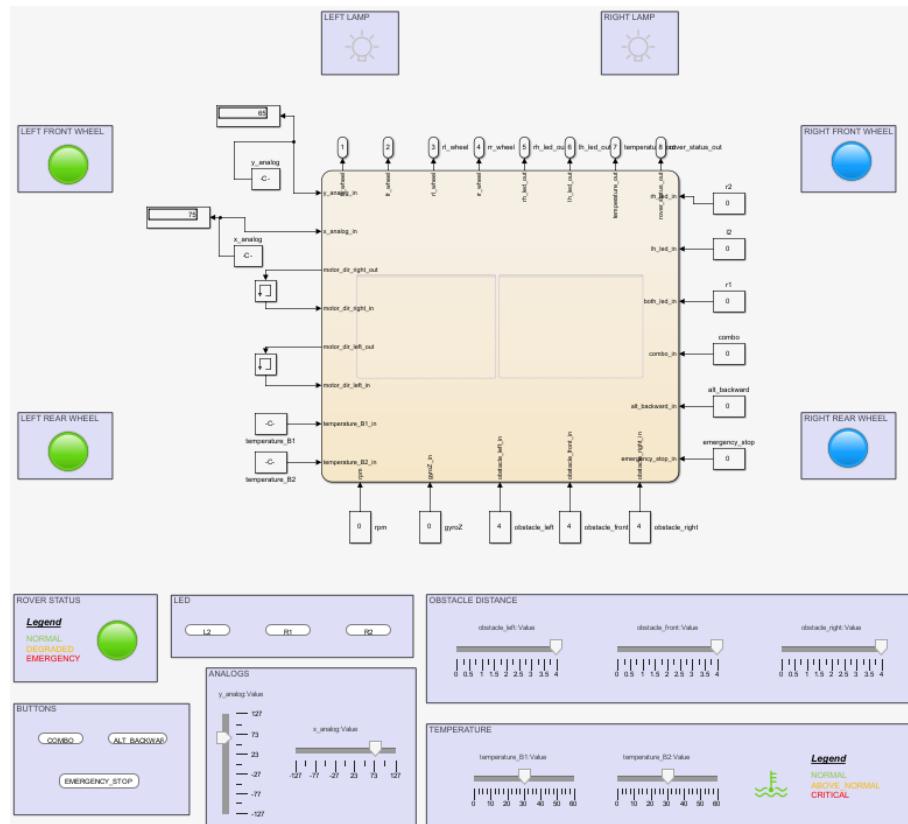


Figura 6.34: Forward right

Successivamente, posizioniamo con un singolo click lo slider *obstacle_right* in modo tale che segni una distanza inferiore a 0.70 m

Macchina a stati

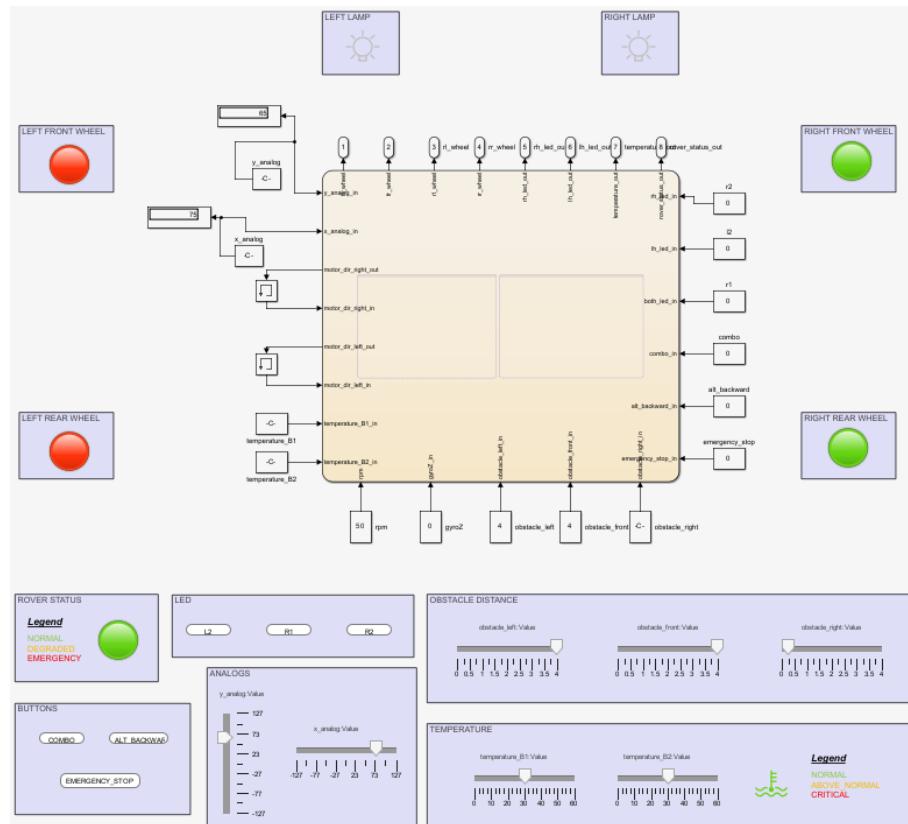


Figura 6.35: Obstacle right

Come è possibile notare, i motori a destra ruotano in senso orario e quelli a sinistra in senso anti-orario, segnalando la rotazione a sinistra del Rover, che termina nel momento in cui questa raggiunge i 90 gradi, arrestando infine i motori.

Macchina a stati

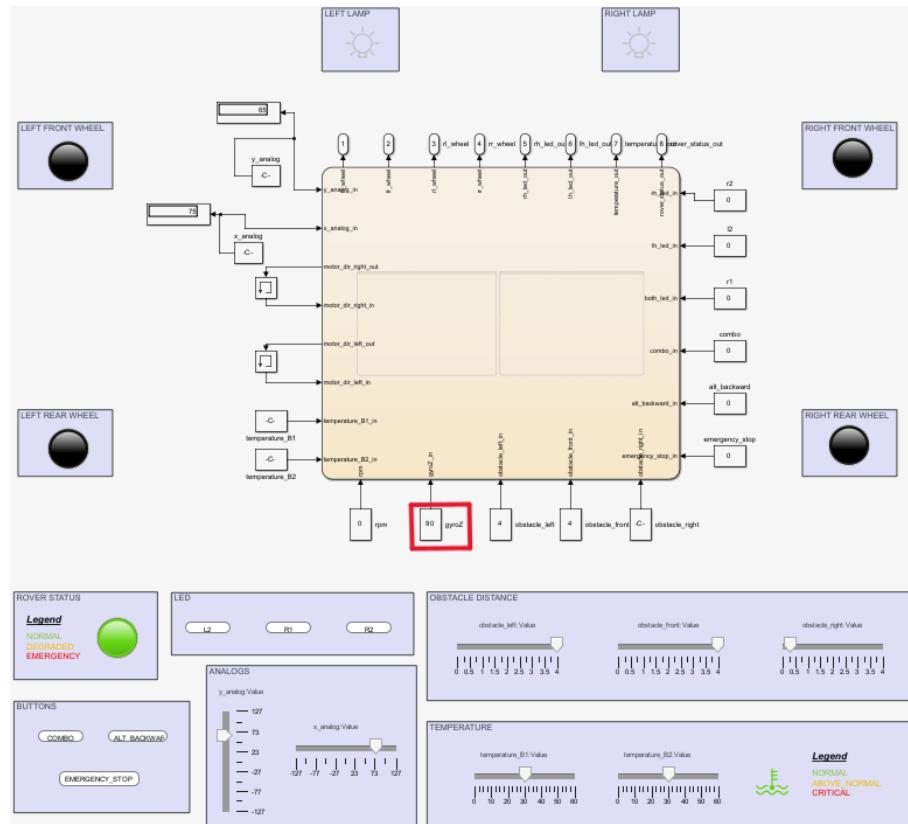


Figura 6.36: Stop

Nota

Si noti come, nonostante gli analogici segnano un valore maggiore di zero, i motori sono fermi data la presenza dell'ostacolo.

6.5.2 Scenario 2: accensione dei led

Partendo dallo stato *FORWARD*

Macchina a stati

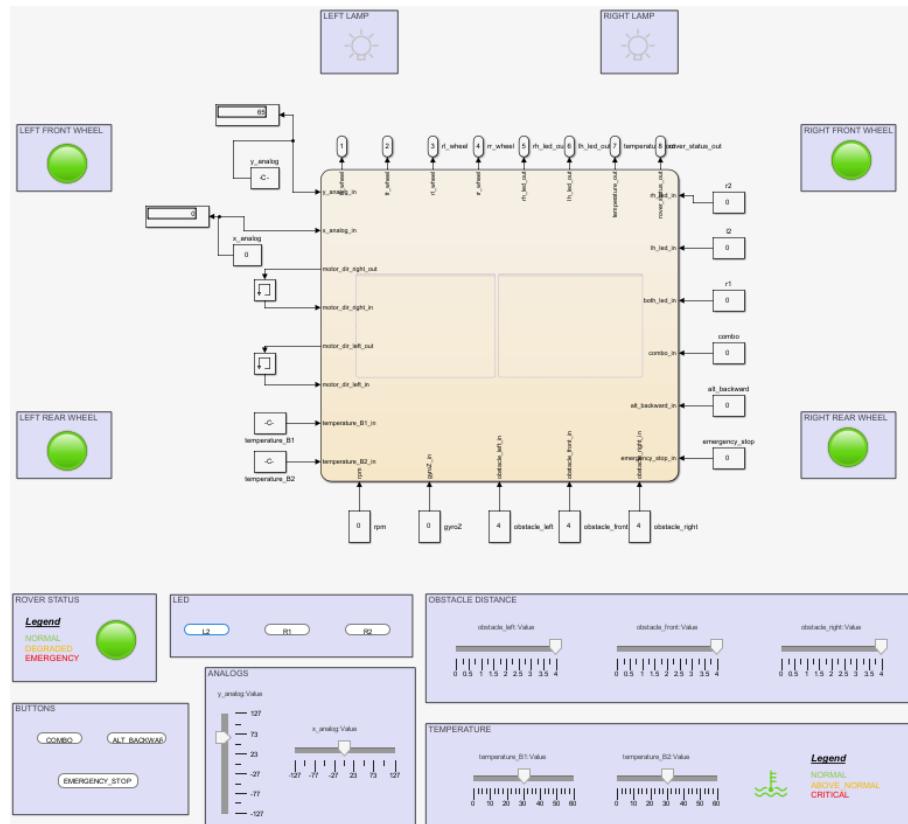


Figura 6.37: Forward

sulla pressione di L2 si ha l'attivazione del led sinistro

Macchina a stati

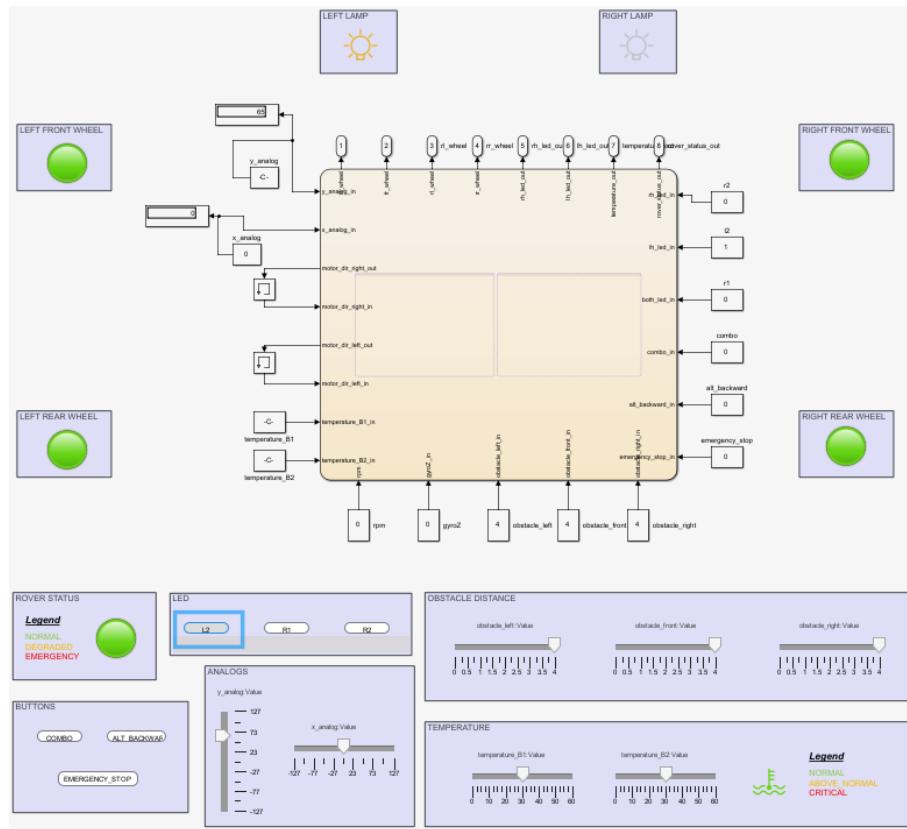


Figura 6.38: Left led on

sulla pressione di R2 si ha l'attivazione del led destro

Macchina a stati

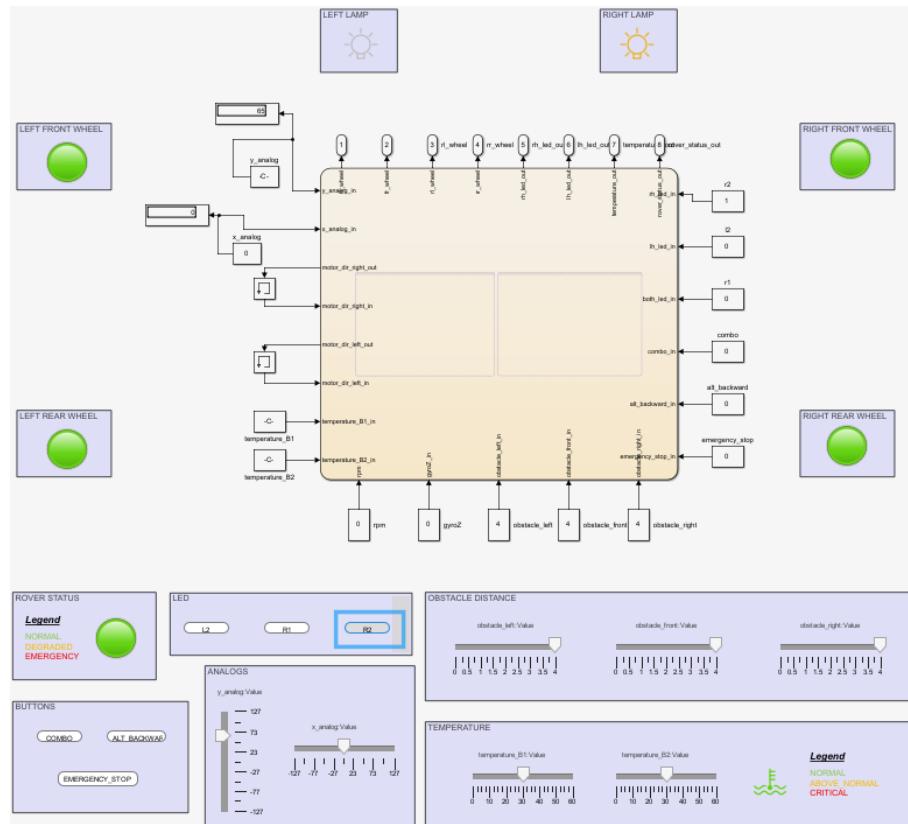


Figura 6.39: Right led on

sulla pressione e rilascio di R1 si ha l'attivazione di entrambi i led

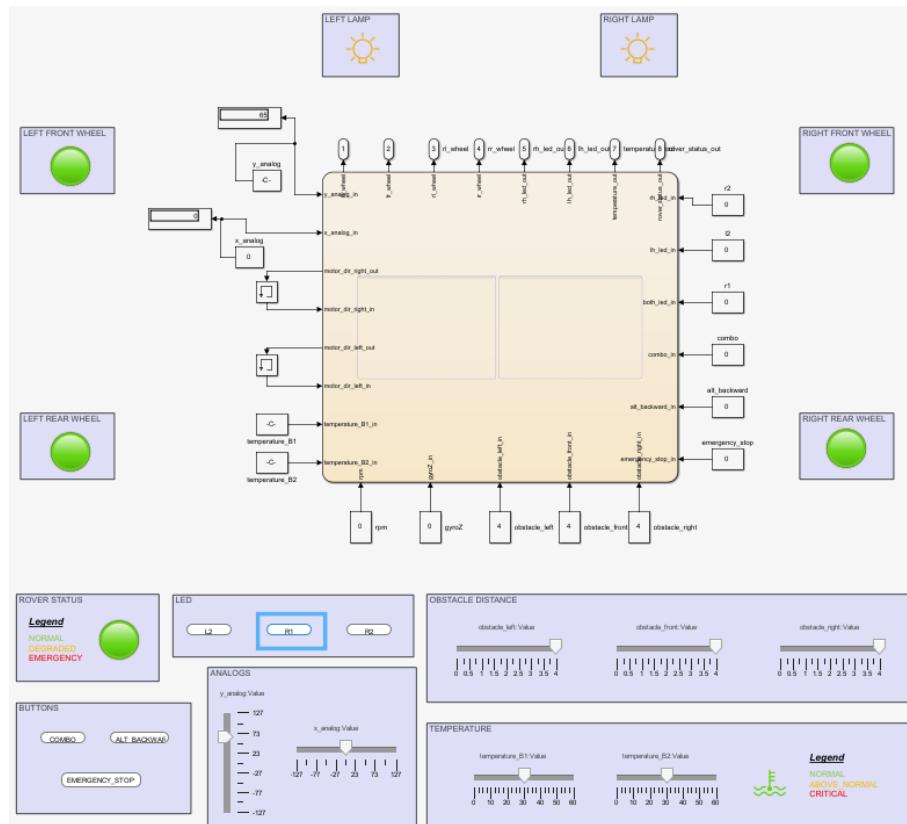


Figura 6.40: Both led on

6.5.3 Scenario 3: aumento della temperatura

Partendo dallo stato FORWARD_LEFT

Macchina a stati

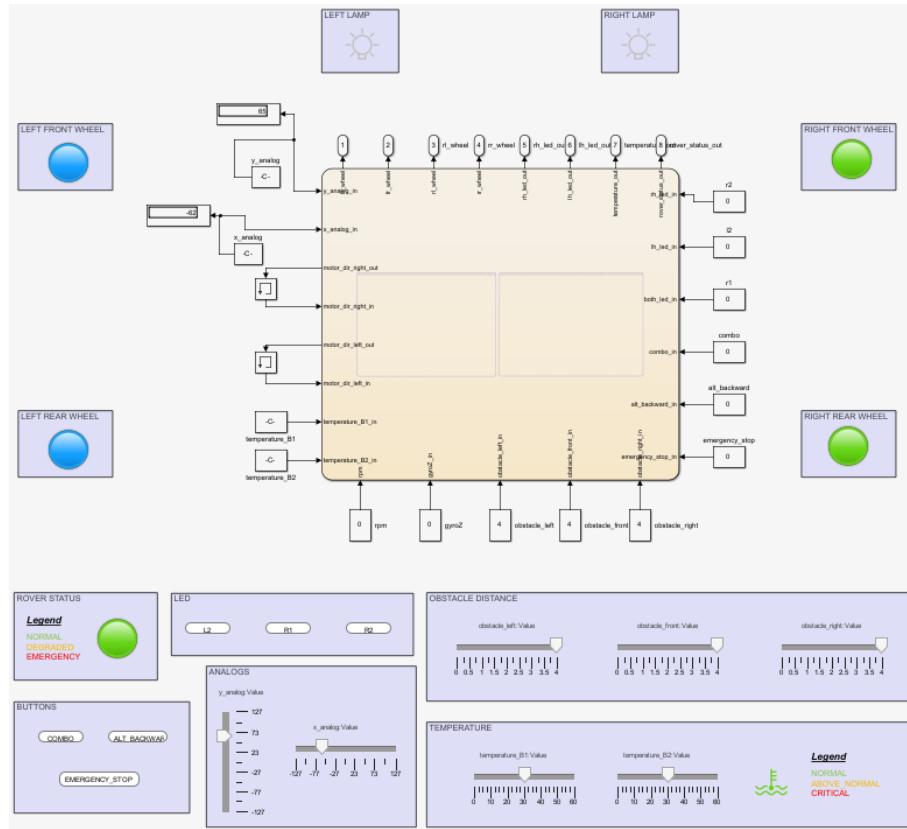


Figura 6.41: Forward left

incrementiamo la temperatura delle due board fino a 38°C utilizzando i relativi slider e notiamo come dopo un certo tempo, posto uguale a 15 secondi nella simulazione effettuata, il sistema continua a funzionare ma in modalità degradata

Macchina a stati

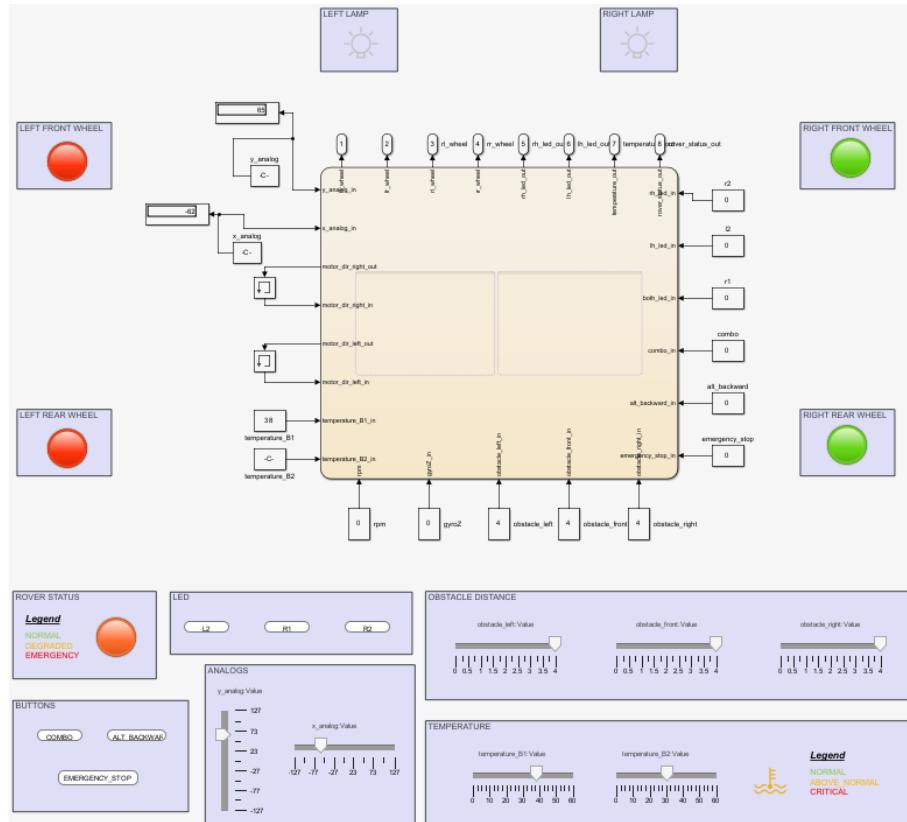


Figura 6.42: Forward left - degraded

Incrementando ulteriormente la temperatura a 58°C, ovvero, al di sopra della soglia critica, il sistema entra in uno stato di emergenza, arrestando i motori

Macchina a stati

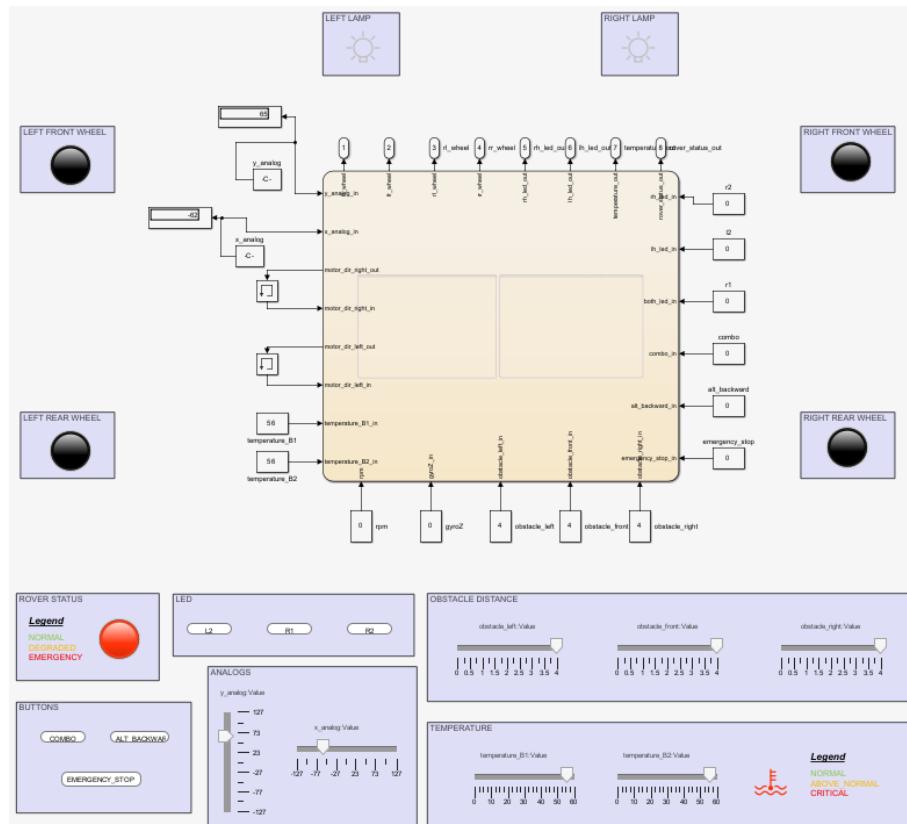


Figura 6.43: Emergency

Capitolo 7

Firmware

In questo capitolo presenteremo brevemente il software e la sua organizzazione soffermandoci sulla struttura del firmware. La directory `Rover` è suddivisa come segue:

```
Rover
└── PrimaryBoard
└── SecondaryBoard
└── CommonDrivers
```

Andremo a trattare brevemente il contenuto di ogni *sub-directory* evidenziandone lo scopo.

7.1 Common Drivers

- **b2b_common.h**: contiene la definizione della struttura `b2b_protocol_t` atta alla gestione del protocollo di comunicazione fra board insieme alle funzioni di protocollo identiche per entrambe le board
- **delayus.h**: contiene sotto forma di macro la definizione assembly di un ciclo bloccante tarato in modo da garantire un'attesa in microsecondi pari al parametro specificato.
- **encoder.h**:
- **encoder_manager.h**:
- **geometry.h**: definisce strutture dati atte a modellare i concetti di punto bidimensionale e tridimensionale
- **hcsr04.h**: mette a disposizione un'interfaccia per l'interazione con il sensore ad ultrasuoni HCSR04. Tramite il driver è possibile leggere la distanza di un ostacolo dal sensore
- **liquid_crystal_I2C.h**: fornisce un'interfaccia che permette di trasferire una buffer di caratteri all'interno della memoria di un display 2X12
- **mpu60x0.h**: fornisce un'interfaccia che permette di interagire con l'accelerometro/giroscopio MPU6050 ottenendo informazioni sull'accelerazione e velocità lungo gli assi x,y,z

- **ps2x.h**: fornisce un’interfaccia che permette di interagire con il controller PS2 ottenendo informazioni circa lo stato degli analogici e la pressione dei tasti
 - **sabertooth_2x12.h**: fornisce un’interfaccia che permette di interagire con un driver sabertooth_2x12 tramite cui è possibile pilotare fino a 2 motori
 - **smoothed_ps2x_analog.h**: implementa le logiche necessarie alla conversione dei valori letti dal controller ps2 a valori di potenza che possono essere forniti al sabertooth_2x12. Tramite questo approccio è possibile imporre una “dinamica” ai motori
 - **state_manager.h**: definisce la struttura `rover_state_t` che contiene tutte le variabili di interesse per il funzionamento dell’intero sistema Rover

7.2 Primary Board

Implementa il firmware necessario al funzionamento della *Board_2* che funge da master per il protocollo *b2b*. Si interfaccia inoltre con i principali sensori, quali:

- PS2x
 - MPU6050
 - Sabertooth 2x12
 - HCSR04

Si occupa dell'aggiornamento delle variabili in Tabella 4.4.

7.2.1 Pinout

In Figura 7.1 si riporta la configurazione relativa alla Board_2

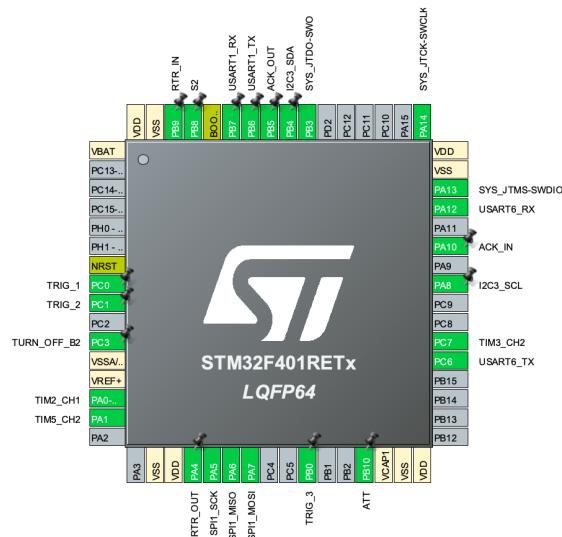


Figura 7.1: Configurazione della Board_2

7.3 Secondary Board

Implementa il firmware necessario al funzionamento della *Board_1* che funge da slave per il protocollo *b2b*. Si interfaccia inoltre con gli encoder associati ai diversi motori e si occupa dell'aggiornamento delle variabili in Tabella 4.5

7.3.1 Pinout

In Figura 7.2 si riporta la configurazione relativa alla *Board_1*.

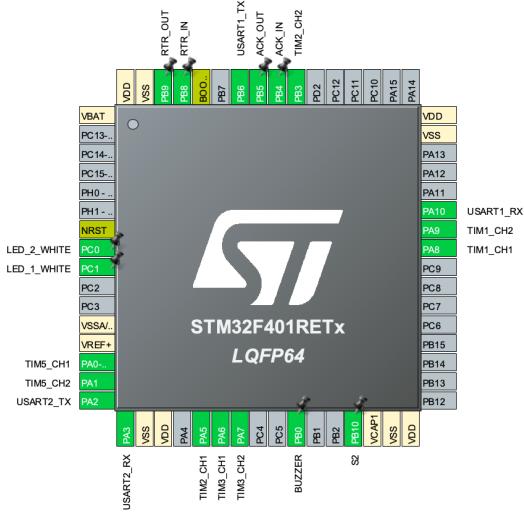


Figura 7.2: Configurazione della board_2

7.4 Single Board

Il progetto *ROVER-FREE_RTOS* presente in *TESTING_PROJECTS* implementa il firmware necessario al funzionamento del rover su singola board. Tale firmware è strutturato per operare in un ambiente FreeRTOS, sfruttando il multitasking per gestire simultaneamente più sensori e attuatori con diverse priorità e frequenze di aggiornamento. Questo approccio modulare e scalabile garantisce un'elevata efficienza e reattività del sistema. Di seguito sono descritte le caratteristiche principali del firmware.

7.4.1 Caratteristiche Principali

Pinout

In Figura 7.3 si riporta la configurazione relativa al firmare con board singola.

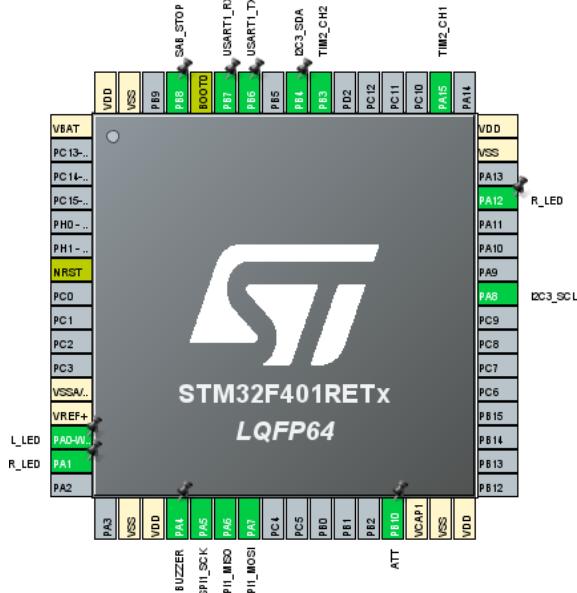


Figura 7.3: Configurazione del Single Board firmware

Modularità e Segmentazione

Il codice è altamente segmentato in task dedicati, ciascuno responsabile di gestire specifiche funzionalità hardware del rover, come la lettura dei dati dal controller PS2x, il controllo dei motori tramite i driver Sabertooth 2x12, la lettura dei valori dal giroscopio e accelerometro MPU6050, e così via. Questa segmentazione facilita la comprensione, la manutenzione e l'eventuale espansione del firmware.

Bilanciamento del Carico di Lavoro

Il firmware sfrutta la capacità di FreeRTOS di bilanciare il carico di lavoro tra i vari task, assegnando priorità diverse in base all'importanza e alla frequenza di aggiornamento richiesta da ciascuna funzionalità. Task critici come la lettura del controller PS2x e il controllo dei motori hanno una frequenza di aggiornamento elevata (25 ms), mentre funzioni meno critiche come la lettura della temperatura o l'aggiornamento del display LCD sono aggiornate con minore frequenza (1000 ms per la temperatura e per l'LCD).

Configurazione Dinamica

Il firmware permette una configurazione dinamica di alcuni parametri, come la frequenza di aggiornamento dei task e l'abilitazione di specifiche funzionalità (es. #define USE_PS2X, #define USE_LCD, etc.), consentendo una facile personalizzazione del comportamento del rover in base alle necessità specifiche.

Gestione degli Errori

Ogni task controlla attivamente il successo delle operazioni eseguite, ricorrendo alla funzione `Error_Handler()` in caso di malfunzionamenti, garantendo così una maggiore robustezza del sistema.

Interfaccia Utente Dinamica

La presenza di task dedicati all'aggiornamento dell'LCD e al controllo delle luci e dei buzzer indica un'attenzione anche verso l'interazione con l'utente, rendendo il rover non solo funzionale ma anche interattivo.

Capitolo 8

Appendice A

8.1 Prototipazione del supporto HCSR04

Per la realizzazione del supporto destinato a ospitare i tre sensori HCSR04, è stata adottata un'approccio di progettazione su misura mediante la creazione di un modello 3D. Questa decisione è stata presa al fine di garantire un adattamento ottimale alle specifiche esigenze dell'applicazione in questione. Il supporto deve essere in grado di ospitare 3 sensori HCSR04, la disposizione ottimale prevede l'utilizzo di un sensore centrale con gli altri due posti a 45° rispetto a quest'ultimo.

8.1.1 Modellazione

Il supporto è stato modellato tramite il software di modellazione 3D Shapr. Il modello è stato basato sulle dimensioni riportate in figura 8.1.

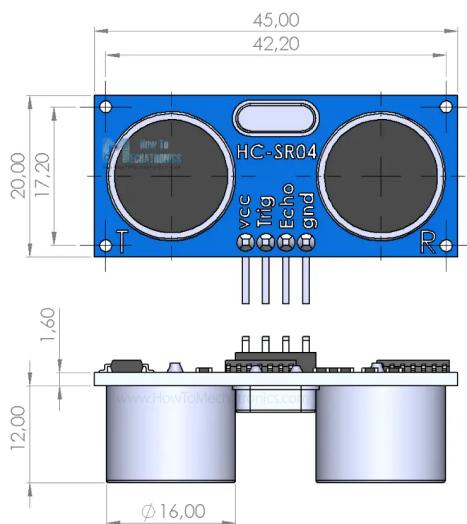


Figura 8.1: Dimensioni di un sensore HCSR04

Il primo elemento realizzato è stata la parte frontale, pensata per contenere i

Appendice A

due lobi cilindrici del sensore ad incastro. È stato considerato un margine di 1 mm su tutte le misure, per coprire eventuali imprecisioni dovute alla stampa.



Figura 8.2: Parte frontale del supporto

Si è proceduto poi a duplicare tale modello in modo da ottenere tre supporti orientati come precedentemente specificato.



Figura 8.3: Parte frontale replicata

Per garantire robustezza al modello e permettere un comodo montaggio sulla parte superiore del rover, si è proceduto nell'aggiunta di una fascia di supporto munita al centro di un piccolo supporto forato, in modo da permettere l'ancoraggio sulla parte superiore del Rover.

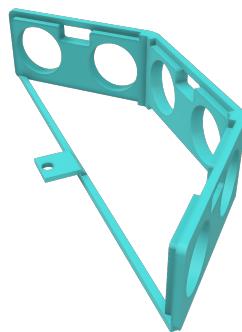


Figura 8.4: Supporto per il montaggio

Successivamente, si è proceduto con la fase di prototipazione mediante l'utilizzo di una stampante 3D. Questo passaggio ha consentito di trasformare il modello virtuale in un oggetto fisico, pronto per essere integrato nel sistema Rover.

Capitolo 9

Pinout e Schema di collegamento

Pinout e Schema di collegamento

Pinout Rover					
PrimaryBoard					
Componente	Pinout				
	<i>ECHO</i>			<i>TRIGGER</i>	
HC-SR04	TIM2_CH1->PA0			PC0	
HC-SR04	TIM5_CH2->PA1			PC1	
HC-SR04	TIM3_CH2->PC7			PB0	
	<i>DAT</i>		<i>CMD</i>		<i>ATT</i> <i>CLK</i>
ps2x	SPI1_MISO->PA6		SPI1_MOSI->PA7		PB10 SPI1_SCK->PA5
	<i>S1 (RX)</i>			<i>S2</i>	
sabertooth 2x12 (anteriore)	USART1_TX->PB6			PB8 (Relay)	
sabertooth 2x12 (posteriore)	USART1_RX->PB6			PB8 (Relay)	
	<i>SDA</i>			<i>SCL</i>	
MPU6050	I2C3_SDA->PB4			I2C3_SCL->PA8	
	<i>TX</i>	<i>RX</i>	<i>ACK_IN</i>	<i>RTR_IN</i>	<i>ACK_OUT</i> <i>RTR_OUT</i>
Board_2	USART6_TX->PC6	USART6_RX->PA12	PA10	PB9	PB5 PA4
				<i>S</i>	PC3
SecondaryBoard					
Componente	Pinout				
	<i>S2</i>				
sabertooth 2x12 (anteriore)	PB10 (Relay)				
sabertooth 2x12 (posteriore)	PB10 (Relay)				
	<i>COMMAND PIN</i>				
Led 1 (sx)	PC1				
Led 2 (dx)	PC0				
	<i>COMMAND PIN</i>				
Active buzzer	PB0				
	<i>A</i>			<i>B</i>	
Motor 1	TIM1_CH1->PA8			TIM1_CH2->PA9	
Motor 2	TIM2_CH1->PA5			TIM2_CH2->PB3	
Motor 3	TIM3_CH1->PA6			TIM3_CH2->PA7	
Motor 4	TIM5_CH1->PA0			TIM5_CH1->PA1	
	<i>TX</i>	<i>RX</i>	<i>ACK_IN</i>	<i>RTR_IN</i>	<i>ACK_OUT</i> <i>RTR_OUT</i>
PrimaryBoard	USART2_TX->PA2	USART2_RX->PA3	PB4	PB8	PB5 PB9

Nella seguente sezione vengono mostrate le scelte fatte sui pin utilizzati su ambedue le MCU, *Primary Board* e *Secondary Board*, con rispettivo schema di collegamento realizzato, riportato nella pagina successiva.

Pinout e Schema di collegamento

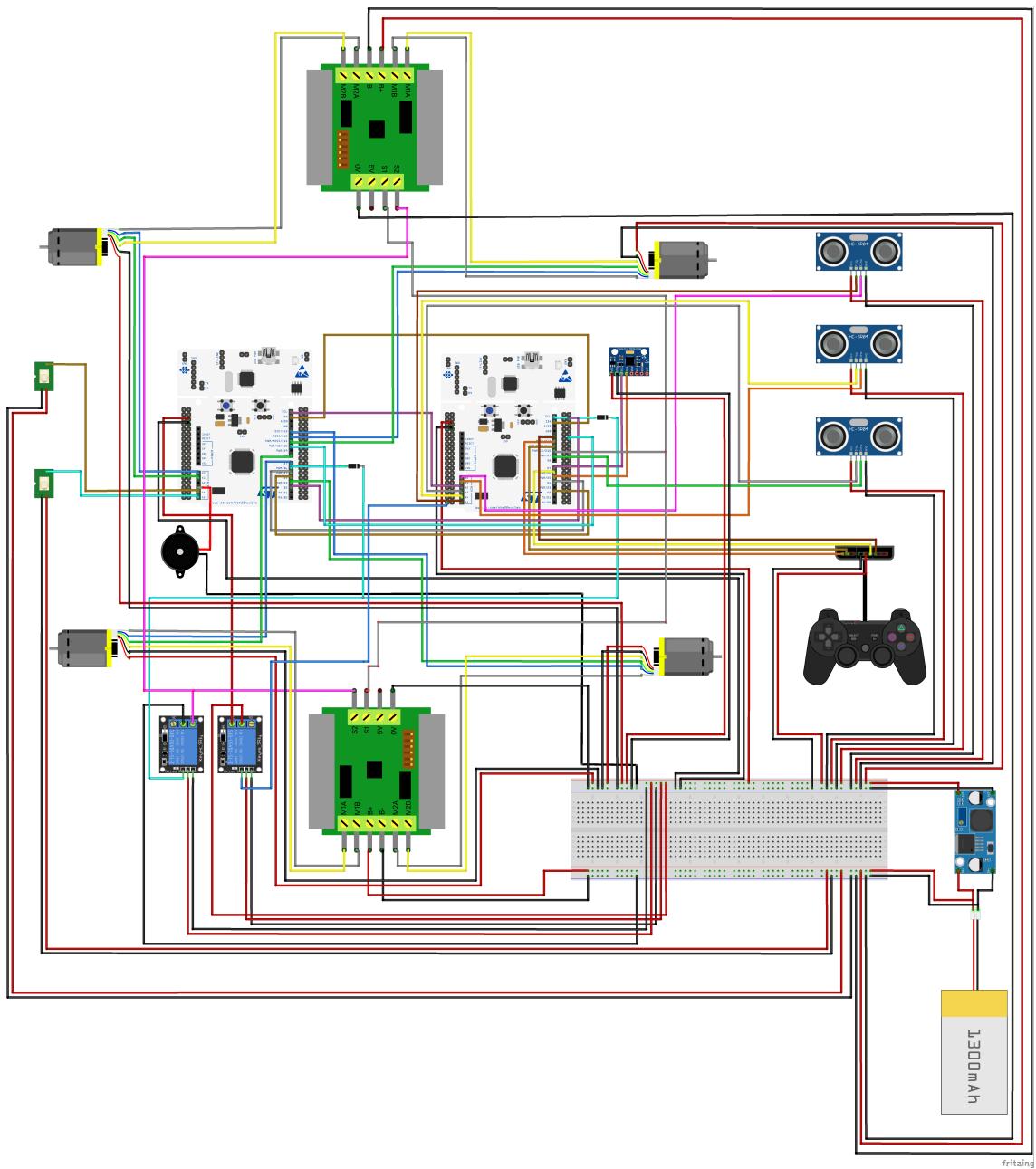


Figura 9.1: Enter Caption

Elenco delle figure

1.1	Architettura di alto livello del sistema	3
3.1	Scenario di inizializzazione del sistema Rover	10
3.2	Scenario di avanzamento del sistema Rover	11
3.3	Scenario di arretramento standard del sistema Rover	13
3.4	Scenario di arretramento alternativo del sistema Rover	15
3.5	Scenario di rotazione del sistema Rover	17
3.6	Scenario di sterzata del sistema Rover	19
3.7	Scenario di avanzamento da fermo con ostacoli del sistema Rover	21
3.8	Scenario di avanzamento con comparsa di ostacoli del sistema Rover	23
3.9	Scenario di spostamento a singolo comando del sistema Rover . .	27
3.10	Scenario di spostamento a comandi combinati del sistema Rover .	29
3.11	Scenario di raggiungimento stato di emergenza del sistema Rover	33
4.1	Tabella delle variabili: nome, tipo e dimensione in bit	35
4.2	Significato dei bit per la variabile <code>control_buttons</code>	36
4.3	Significato dei bit per la variabile <code>control_values</code>	36
4.4	Tabella delle variabili il cui aggiornamento è sotto la responsabilità della <code>board_1</code>	36
4.5	Tabella delle variabili aggiornate dalla <code>board_2</code>	37
4.6	Tabella delle variabili codificant i l'intenzione di ognuna delle due <code>board</code>	37
5.1	Sensore HCSR04	39
5.2	Tempi di risposta del sensore HCSR04 al variare della distanza di un ostacolo	40
5.3	Tempi di risposta della funzione <code>read_gamepad_analog</code>	41
5.4	Comportamento della <code>HAL_I2C_MemRead</code>	42
5.5	"Transfer sequence diagram for master receiver" - STM32F401x reference manual	42
5.6	Tempi di risposta della funzione <code>MPU60X0_get_gyro_value</code>	43
5.7	Tempi di risposta della funzione <code>encoder_manager_update_rpm</code> .	43
5.8	Vista di alto livello del protocollo di comunicazione fra <code>board</code> . .	45
5.9	Trasmissione dello stato Board_1 → Board_2 misurata: 2.6 ms . .	46
5.10	Trasmissione dello stato Board_2 → Board_1 misurata: 3.02 ms . .	46
5.11	Protocollo Board to Board.	47
5.12	Tempo impiegato: 16.64 ms	49

5.13 Esempio di scheduling dove il task di lettura dei sensori ha un periodo sensibilmente inferiore rispetto a quello di comunicazione.	52
5.14 Scheduling previsto	55
5.15 Effetti del <i>clock drifting</i> su due clock identici con la stessa frequenza di oscillazione	57
5.16 Scheduling finale ottenuto	59
6.1 Rover chart	60
6.2 BOARD_1	63
6.3 Initialization	68
6.4 Working	69
6.5 State updating	70
6.6 Communication and decision	70
6.7 Communication	71
6.8 Waiting BOARD_2	71
6.9 Temperature controller	73
6.10 <code>check_temperature</code> function	74
6.11 led control functions	74
6.12 Led controller	75
6.13 BOARD_2	76
6.14 Initialization	78
6.15 Working	79
6.16 State updating	79
6.17 Communication and decision	80
6.18 Communication	81
6.19 Action	82
6.20 Single board	83
6.21 Execution	84
6.22 Move and stop functions	86
6.23 Board_2 - Emergency	86
6.24 Board_1 - Emergency	87
6.25 Modified chart for simulation	88
6.26 Rover status	89
6.27 Led button	89
6.28 Buttons for other functionality	90
6.29 Analogs slider	90
6.30 Obstacle distance slider	90
6.31 Temperature slider	91
6.33 Idle	92
6.34 Forward right	93
6.35 Obstacle right	94
6.36 Stop	95
6.37 Forward	96
6.38 Left led on	97
6.39 Right led on	98
6.40 Both led on	99
6.41 Forward left	100

6.42 Forward left - degraded	101
6.43 Emergency	102
7.1 Configurazione della Board_2	104
7.2 Configurazione della board_2	105
7.3 Configurazione del Single Board firmware	106
8.1 Dimensioni di un sensore HCSR04	108
8.2 Parte frontale del supporto	109
8.3 Parte frontale replicata	109
8.4 Supporto per il montaggio	110
9.1 Enter Caption	113