

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INGEGNERIA INFORMATICA ED ELETTRICA
E MATEMATICA APPLICATA



Laurea Magistrale in Ingegneria Informatica
Gruppo 3: Documentazione Embedded System Design

Nome	Cognome	Matricola	Email
Alessandro	Pentangelo	0622702632	a.pentangelo18@studenti.unisa.it
Mirko	Campanella	0622702588	m.campanella1@studenti.unisa.it
Michele	Colombo	0622702472	m.colombo1@studenti.unisa.it
Alessio	Bottiglieri	0622702583	a.bottiglieri16@studenti.unisa.it

Contents

1	Introduzione work-project	5
1.1	Architettura Hardware e Comunicazione	5
1.2	Dinamica di Movimento e Controllo	6
1.3	Sicurezza e Gestione degli Ostacoli	6
1.4	Resilienza e Modalità Degradata	6
2	Sensoristica del sistema	8
2.1	Sensori Board 1	8
2.2	Sensori Board 2	8
3	Elicitazione ed Analisi dei requisiti	9
3.1	Requisiti funzionali di alto livello	9
3.2	Requisiti su malfunzionamenti e soglie di allarme	20
3.3	Requisiti di modellazione e progettazione del controllo	24
3.4	Requisiti di schedulazione	28
3.5	Parametri scelti in fase di progettazione	30
4	Funzionamento generale del software di ciascuna board	33
4.1	Interazione tra Task e Meccanismo di Snapshot	33
4.2	Funzionamento task di Board 2	35
4.2.1	Task: acquisizione distanza dagli ostacoli	35
	Utilizzo DMA per la lettura dei segnali	36
4.2.2	Task: lettura comandi utente	37
	Interfacciamento DualSense PS5 tramite ESP32	39
4.2.3	Task: lettura giroscopio	42
4.2.4	Task: supervisore Board 2	43
4.2.5	Task: log dei dati per il debug	47
4.3	Funzionamento task di Board 1	48
4.3.1	Task: lettura batteria e temperatura	48
4.3.2	Task: supervisore Board 1	50
4.3.3	Task: controllo motori	53
	Struttura del PI gerarchico del Rover	53
	Controllo globale della velocità	54
	Sincronizzazione tra asse anteriore e asse posteriore	54
	Controllo differenziale e gestione della rotazione	55
	Sistema di attuazione	56
	Modalità di controllo dei driver	56
	Struttura del pacchetto seriale	58

4.3.4	Task: log dei dati per il debug	59
4.3.5	Task: led	60
4.4	Task di trasmissione, ricezione e protocollo di comunicazione tra le due board	61
4.4.1	Scelta di comunicazione asincrona	61
4.4.2	Ricostruzione dello stato globale tramite lo scambio dati	62
4.4.3	Struttura del protocollo di comunicazione	64
	Sincronizzazione del flusso dati	65
	Gestione dei fault di comunicazione	66
5	Algoritmo di schedulazione dei task	67
5.1	Calcolo del Fattore di Utilizzo	67
5.2	Criterio di Schedulabilità: Limite di Liu-Layland	67
5.3	Verifica Numerica dei task su Board 1	68
5.4	Verifica Numerica dei task su Board 2	68
5.5	Assegnazione delle priorità	69
6	Gestione della concorrenza e sincronizzazione: i Mutex	70
	Perché utilizzare i Mutex	70
6.1	Snapshot come struttura condivisa	70
6.1.1	Protezione degli snapshot tramite mutex	71
6.1.2	Mutex e snapshot implementati su Board 1	74
6.2	Mutex e snapshot implementati su Board 2	75
7	Condizioni di funzionamento	77
7.1	Condizioni di funzionamento nominale	77
	Rilevamento ostacoli	78
7.2	Modalità operative non nominali	78
	Modalità degradata con entrambe le Board operative .	78
	Modalità degradata con Board isolata	78
	Modalità critica	79
7.2.1	Esempio di funzionamento in modalità non nominale: fault e fallback encoder	79
8	Supervisore Board 1	82
8.1	Panoramica generale	82
8.2	Ingressi del supervisore	83
8.3	Uscite del supervisore	85
8.4	Rilevazione Faults locali	86
8.4.1	Esempio: monitoraggio della ricezione dalla Board 2 .	88
8.4.2	Output del monitoraggio	90

Costruzione delle maschere critiche e degradate	92
8.5 Selezione della board attuante	92
8.6 Stato del rover e calcolo dei riferimenti (solo quando attua Board 1)	94
8.6.1 Struttura interna del sottosistema di decisione	95
Check rover safety state	96
Rover motion state	96
Backward_mode_toggle	97
Rover motion state	98
9 Supervisore Board 2	101
9.1 Panoramica generale	101
9.2 Ingressi del supervisore	102
9.3 Uscite del supervisore	103
9.4 Rilevazione Faults	105
MonitorBLE	106
MonitorBoard1Supervisor	106
MonitorIMU	107
MonitorRx	107
9.5 Costruzione maschere degradate e critiche	108
9.6 Logica che consente a Board 2 di attuare	109
9.7 Rilevamento ostacoli	111
9.7.1 Gestione ostacoli con Board 1 attuatrice	112
9.7.2 Gestione ostacoli con Board 2 attuatrice	115
9.8 Attuazione	116

1 Introduzione work-project

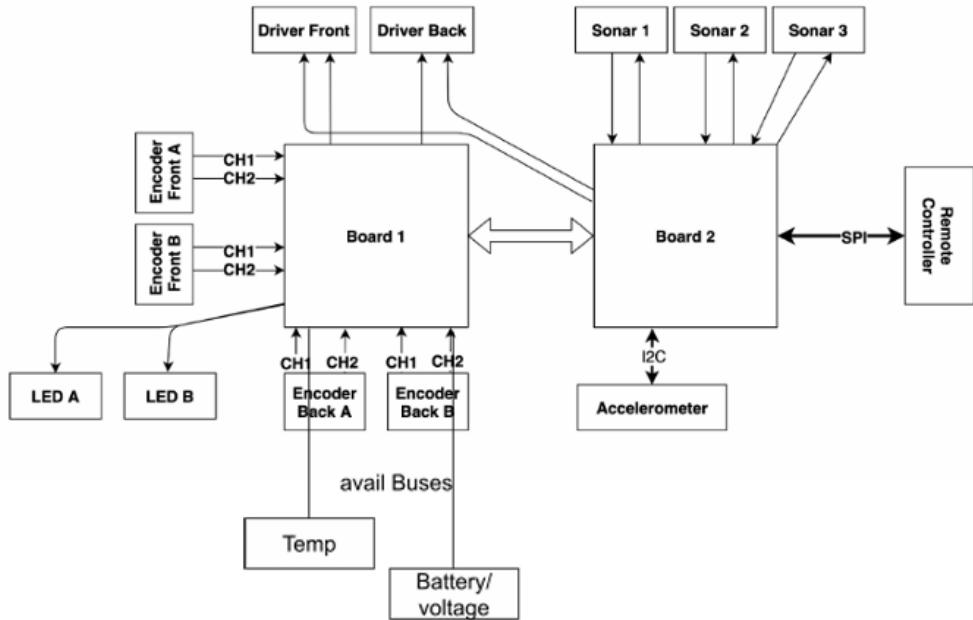


Figure 1: Architettura generale del rover distribuita su doppia board

Il presente progetto riguarda la progettazione e l'implementazione di un **sistema di controllo e supervisione distribuito** per un rover caratterizzato da un'architettura ad alta affidabilità. L'obiettivo principale è lo sviluppo di un veicolo capace di muoversi in modo coordinato, reagire prontamente agli stimoli ambientali rilevati tramite sensoristica avanzata e gestire scenari di guasto attraverso logiche di ridondanza e modalità di funzionamento degradate.

1.1 Architettura Hardware e Comunicazione

Il cuore del sistema è costituito da un'architettura a **doppia scheda** (Board 1 e Board 2), basata su microcontrollori STM32. Questa configurazione implementa il concetto di **diversità di livello 2**, dove le due board collaborano per la ricostruzione dello stato globale del sistema pur operando su partizioni di dati e algoritmi distinti.

- La Board 1 gestisce l’interfacciamento con i driver dei motori (Front e Back), gli encoder di posizione e i LED di segnalazione.
- La Board 2 è dedicata alla sensoristica ambientale, interfacciandosi con tre sensori Sonar, un accelerometro tramite protocollo I^2C e il sistema di controllo remoto, che avviene tramite l’utilizzo di un joystick che invia dati a una scheda ESP32 tramite protocollo BLE.

1.2 Dinamica di Movimento e Controllo

Il rover deve garantire una movimentazione precisa sincronizzando i quattro motori per procedere in linea retta o ruotare attorno al proprio baricentro.

- **Controllo PID:** La velocità richiesta dal controller viene gestita tramite un algoritmo PID software per definire le dinamiche di raggiungimento del target e di frenata.
- **Attuazione:** Una volta calcolata l’uscita di controllo tramite il PID, questa viene mandata in attuazione ai motori tramite il driver saber-tooth configurato in modalità Serial Packetized.

1.3 Sicurezza e Gestione degli Ostacoli

Il sistema di supervisione utilizza i dati provenienti dai sensori ad ultrasuoni per garantire l’integrità del veicolo:

- **Rilevamento Preventivo:** Identificazione di ostacoli fino a 3 metri di distanza con un angolo di copertura di 45°.
- **Manovre di Emergenza:** In caso di ostacoli improvvisi a meno di 70 cm, il rover esegue una frenata d’emergenza seguita da una rotazione verso un lato libero.
- **Evitamento Dinamico:** Capacità di evitare ostacoli in movimento tra 1,5 e 3 metri sterzando senza interrompere la marcia.

1.4 Resilienza e Modalità Degradata

Una specifica fondamentale del progetto è la capacità di operare in presenza di malfunzionamenti. Il sistema monitora costantemente lo stato delle board,

dei motori, della batteria e della temperatura interna. In caso di determinati fallimenti, il rover entra in **Modalità Degradata** o in **Emergency Stop**.

2 Sensoristica del sistema

2.1 Sensori Board 1

La Board 1 è responsabile del loop di controllo dei motori, della gestione dei led presenti sul rover e del monitoraggio della batteria e della temperatura interna del rover (Board Health). I sensori e gli attuatori gestiti direttamente da questa board includono:

- **Encoder incrementali:** Quattro encoder (Front/Back, SX/DX) per il feedback sulla velocità reale delle ruote.
- **Driver Motori:** Due driver sabertooth 2x12, uno che si occupa di controllare le ruote anteriori e un altro che si occupa del controllo delle ruote posteriori. Entrambi i driver sono configurati in modalità Serial Packetized.
- **Sensore di Tensione e Temperatura:** Sensori per il monitoraggio dello stato della batteria e della temperatura.
- **LED di stato:** Due indicatori luminosi (LED A e LED B) per il feedback visivo immediato (es. stato di emergenza).

2.2 Sensori Board 2

La Board 2 funge da interfaccia di input del Rover. Il suo compito principale è raccogliere i dati dall'ambiente esterno e dai comandi dell'operatore, elaborarli e trasmetterli alla Board 1 per l'attuazione fisica. I componenti e i sensori gestiti da questa board sono:

- **Modulo BLE (Bluetooth Low Energy):** Per la ricezione dei pacchetti dati dal controller remoto (Direzione, Velocità, Modalità).
- **IMU (Inertial Measurement Unit):** Accelerometro e giroscopio per il monitoraggio dell'assetto e della stabilità del Rover.
- **Sonar (Ultrasuoni):** Tre sensori posizionati strategicamente per la navigazione ed il superamento degli ostacoli fino a 3 metri di distanza.

3 Elicitazione ed Analisi dei requisiti

Questa sezione descrive il processo di elicitatione e analisi dei requisiti del sistema, a partire dal documento di specifica fornito per il Work Project. I requisiti rappresentano le funzionalità attese, i vincoli operativi e i criteri di sicurezza che il rover deve soddisfare.

L'attività di analisi ha lo scopo di tradurre le specifiche di alto livello in requisiti chiari, verificabili e utilizzabili come base per la progettazione del sistema embedded, della logica di supervisione e del controllo del rover. Quando necessario, sono state effettuate scelte progettuali per definire in modo preciso il comportamento del sistema, nel rispetto delle specifiche fornite.

I requisiti sono organizzati nelle seguenti categorie:

- requisiti funzionali di alto livello, che descrivono le principali funzionalità del sistema;
- requisiti su malfunzionamenti e soglie di allarme, che definiscono la condizione di malfunzionamento ed il comportamento che il sistema deve adottare in tali condizioni
- requisiti di modellazione e progettazione del controllo, che definiscono la rappresentazione dello stato del sistema, la logica di supervisione e i criteri adottati per la generazione e gestione dei riferimenti di controllo;
- requisiti di schedulazione, che descrivono i vincoli temporali e l'organizzazione dei task software;
- parametri scelti in fase di progettazione, che raccolgono i valori numerici e le configurazioni definite per l'implementazione del sistema.

Per ogni requisito funzionale vengono inoltre riportate una User Story e i relativi Acceptance Criteria, che ne chiariscono lo scopo e permettono di verificarne la corretta implementazione.

3.1 Requisiti funzionali di alto livello

R1. Comandi esterni BLE

Requisito: Il Rover deve accettare comandi esterni da un controller remoto wireless tramite protocollo BLE.

User Story: come operatore, voglio comandare il rover da controller BLE, così da guidarlo da remoto.

Acceptance Criteria:

- Dato il rover acceso, quando il controller BLE invia un comando valido, allora il comando viene acquisito e reso disponibile al controllo.
- Dato un pacchetto BLE non valido/corrotto, quando viene ricevuto, allora non modifica il setpoint di controllo e viene segnalato come errore di comunicazione.

R2. Modalità di invio comandi (SCELTA PROGETTUALE)

Requisito: I comandi sono inviati dal controller in modalità analogica, tra le alternative previste dalla specifica.

User Story: come progettista software, voglio utilizzare input analogici dalle levette del controller, così da ottenere un comando continuo e proporzionale alla posizione della levetta.

Acceptance Criteria:

- Dato il controller configurato in modalità analogica, quando viene inviato un input valido, allora il rover interpreta il comando in modo continuo e proporzionale.
- Dato un input non valido o fuori dal range previsto, quando viene ricevuto, allora il sistema non lo utilizza per aggiornare il setpoint.

R3. Interpretazione levette (SCELTA PROGETTUALE)

Requisito: La posizione della levetta è interpretata con semantica diretta proporzionale, ovvero l'inclinazione determina direttamente il setpoint istantaneo di velocità.

User Story: come operatore, voglio che la posizione della levetta determini direttamente la velocità del rover, così da avere un controllo immediato e intuitivo del movimento.

Acceptance Criteria:

- Dato un asse levetta normalizzato $u \in [-1, 1]$, quando il comando viene letto, allora il setpoint soddisfa

$$v_{\text{set}} = u \cdot v_{\max}$$

con eventuale saturazione ai limiti ammessi.

- Dato un valore costante della levetta, quando il comando viene mantenuto nel tempo, allora il setpoint resta costante.

R4. Uso combinazioni pulsanti (SCELTA PROGETTUALE)

Requisito: Le combinazioni di pulsanti del controller sono utilizzate per attivare funzioni operative e di sicurezza definite dal sistema.

User Story: come operatore, voglio utilizzare combinazioni di pulsanti per attivare funzioni specifiche, così da poter controllare rapidamente modalità operative e funzioni di sicurezza.

Acceptance Criteria:

- Dato un insieme definito di combinazioni di pulsanti e delle rispettive funzioni associate, quando l'operatore esegue una combinazione valida, allora il sistema attiva la funzione corrispondente.
- Dato una combinazione di pulsanti non associata ad alcuna funzione, quando viene ricevuta, allora il sistema non modifica lo stato operativo.
- Dato una funzione associata a una combinazione di pulsanti, quando viene attivata, allora il sistema entra nello stato operativo previsto.

R5. Marcia rettilinea

Requisito: Il rover deve muoversi in linea retta avanti/indietro alla velocità predefinita, sincronizzando i 4 motori.

User Story: come operatore, voglio che il rover proceda dritto sia in avanti che indietro, così da avere traiettorie stabili in entrambi i versi di marcia.

Acceptance Criteria:

- Dato un riferimento rettilineo in avanti o indietro, quando il rover marcia, allora i quattro motori restano sincronizzati entro la tolleranza definita e la traiettoria resta rettilinea.

R6. Controllo dinamico velocità

Requisito: La velocità richiesta deve essere controllata da PID software per definire dinamica di accelerazione e frenata.

User Story: come progettista controllo, voglio regolare la dinamica con PID, così da ottenere una risposta più smooth, prevedibile e stabile sia in accelerazione sia in frenata.

Acceptance Criteria:

- Dato un gradino di riferimento velocità, quando il controllo PID è attivo, allora la velocità converge al setpoint con dinamica coerente al tuning e senza variazioni brusche non previste.
- Dato un comando di riduzione velocità o arresto, quando il controllo PID è attivo, allora la frenata risulta progressiva.

R7. Rotazione su baricentro

Requisito: Il rover deve ruotare attorno al proprio baricentro sincronizzando i 4 motori.

User Story: come operatore, voglio che il rover giri su se stesso quando do solo input laterale sul joystick (senza input in avanti), così da orientarlo sul posto.

Acceptance Criteria:

- Dato input joystick solo laterale e input avanti nullo, quando il rover esegue la manovra, allora ruota su se stesso con coordinamento dei quattro motori.

R8. Acquisizione sensori

Requisito: Il rover deve leggere i dati dai sensori definiti e interfacciati con i relativi protocolli.

User Story: come supervisore di sistema, voglio acquisire i sensori periodicamente, così da stimare lo stato corrente del rover.

Acceptance Criteria:

- Dato il ciclo di task attivo, quando scatta il periodo di acquisizione, allora gli snapshot sensore vengono aggiornati.

R9. Rilevamento ostacoli fino a 3 m

Requisito: Uso sensori ultrasound per ostacoli almeno a 3 m davanti, 45° sinistra e 45° destra anteriori.

User Story: come operatore, voglio rilevare ostacoli in anticipo frontalmente, così da prevenire collisioni.

Acceptance Criteria:

- Dato un ostacolo nel campo di vista dei tre sensori, quando entra nella fascia misurabile, allora viene segnalato al supervisore.

R10. Stato sicuro con ostacolo fermo

Requisito: Con ostacolo fermo nella direzione di marcia anteriore, il rover deve entrare in "Motori Fermi" prima dell'impatto.

User Story: come responsabile sicurezza, voglio fermata preventiva su ostacolo fermo, così da evitare urti.

Acceptance Criteria:

- Dato un ostacolo fermo rilevato frontalmente, quando la distanza scende sotto la soglia di sicurezza, allora il rover arresta i motori in tempo utile.

R11. Frenata smooth (OPZIONALE)

Requisito: La frenata del requisito R10 puo' essere realizzata con dinamica smooth a risposta al gradino predefinita.

User Story: come progettista controllo, voglio una frenata morbida, così da ridurre stress meccanico.

Acceptance Criteria:

- Dato il profilo smooth attivo, quando si genera comando di stop per ostacolo fermo, allora la velocità segue il profilo di frenata definito.

R12. Emergenza sotto 70 cm

Requisito: Con ostacolo improvviso sotto 70 cm in una delle 3 direzioni, il rover deve frenare di emergenza e, se possibile, ruotare verso lato libero.

User Story: come sistema safety, voglio reagire immediatamente a ostacolo improvviso vicino, così da massimizzare la probabilità di evitare impatto.

Acceptance Criteria:

- Dato ostacolo sotto 70 cm, quando viene confermato dai sensori, allora viene attivata frenata di emergenza.
- Se almeno un lato è libero, viene comandata rotazione verso un lato libero (casuale se entrambi liberi).

R13. Blocco partenza con ostacolo frontale

Requisito: Il rover non deve avanzare da fermo se è presente ostacolo nella direzione di marcia.

User Story: come operatore, voglio che la partenza in avanti sia inibita con ostacolo davanti, così da prevenire collisioni immediate.

Acceptance Criteria:

- Dato rover fermo e ostacolo frontale rilevato, quando viene richiesto avanzamento, allora il comando viene bloccato.
- Dato rover fermo e ostacolo frontale rilevato, quando viene richiesto di ruotare sul posto o di arretrare, allora tali comandi restano consentiti nel rispetto delle regole di sicurezza.

R14. Retromarcia con rotazione 180° (OPZIONALE)

Requisito: Quando è attiva la modalità di retromarcia assistita, il comando di retromarcia da fermo viene realizzato con rotazione di 180° e successiva marcia in avanti se il percorso è libero.

User Story: come operatore, voglio una procedura assistita di retromarcia sicura, così da evitare manovre rischiose.

Acceptance Criteria:

- Dato che la modalità "retromarcia assistita 180°" è attiva, quando l'operatore richiede la retromarcia da fermo, allora il rover esegue una rotazione di 180° e procede solo se la direzione risultante è libera da ostacoli.
- Durante l'esecuzione della manovra assistita, i comandi dell'operatore non hanno effetto fino al completamento della rotazione.
- La presenza di ostacoli nelle direzioni non coinvolte nella traiettoria prevista non interrompe la manovra, mentre eventuali ostacoli nella direzione di marcia impediscono l'avanzamento.
- La manovra dovrà essere interrotta al verificarsi di una condizione critica.

R15. Comutazione modalità retromarcia con combo (OPZIONALE)

Requisito: La selezione della modalità di retromarcia deve avvenire tramite combo dedicata.

User Story: come progettista safety, voglio una combo esplicita per commutare la modalità di retromarcia, così da evitare attivazioni involontarie e ambiguità operative.

Acceptance Criteria:

- Dato il sistema in stato iniziale, quando non è stata eseguita alcuna combo di commutazione, allora la modalità attiva è la retromarcia assistita 180°.
- Dato che è attiva la modalità assistita 180°, quando viene eseguita la combo di commutazione, allora si attiva la retromarcia normale e si disattiva la modalità assistita.
- Dato che è attiva la retromarcia normale, quando viene rieseguita la combo di commutazione, allora si riattiva la modalità assistita 180° e si disattiva la retromarcia normale.

R16. Gestione fault critici

Requisito: Il rover deve gestire i fault critici garantendo il raggiungimento di uno stato sicuro.

User Story: come sistema safety, voglio distinguere i fault critici per applicare la reazione più sicura (arresto o trasferimento dell'attuazione), così da portare il rover in uno stato sicuro anche in presenza di malfunzionamenti.

Acceptance Criteria:

- Dato un fault critico relativo a periferiche o allo stato di salute del rover, quando viene rilevato dal supervisore, allora il sistema porta il rover in uno stato sicuro con motori fermi.
- Dato un fault critico relativo alla comunicazione tra board, quando il trasferimento è possibile, allora l'attuazione viene demandata all'altra board, mantenendo il rover in uno stato sicuro secondo la logica di alta affidabilità.

R17. Marcia degradata su fault specifici (OPZIONALE)

Requisito: Determinate combinazioni di malfunzionamento consentono al rover di operare in modalità degradata con limitazione della velocità massima.

User Story: come operatore, voglio mantenere una funzionalità ridotta in presenza di fault non critici, così da poter completare manovre essenziali in condizioni di sicurezza.

Acceptance Criteria:

- Dato un fault o una combinazione di fault appartenente all'insieme definito per la modalità degradata, quando il rover resta operativo, allora il sistema entra in modalità degradata e applica una limitazione alla velocità massima.
- Dato che il rover opera in modalità degradata, quando vengono impartiti comandi di movimento, allora la velocità del rover è limitata al valore massimo consentito per tale modalità.

R18. Evitamento ostacolo in movimento (1.5–3 m)

Requisito: Il rover deve evitare un ostacolo in movimento rilevato tra 1.5 m e 3 m mediante sterzata nella direzione del primo sensore che lo ha rilevato, dopo conferma da parte di un secondo sensore, senza interrompere la marcia. Se la distanza scende sotto 75 cm, il rover deve fermarsi.

User Story: come operatore, voglio che il rover eviti automaticamente ostacoli in movimento quando rilevati tra 1.5 m e 3 m, così da ridurre il rischio di collisione e mantenere la continuità di marcia, fermandosi comunque se la distanza scende sotto 70 cm.

Acceptance Criteria:

- Dato un ostacolo rilevato sequenzialmente da due sensori entro una finestra temporale definita e a distanza compresa tra 1.5 m e 3 m, quando il secondo sensore conferma la rilevazione, allora il rover comanda una sterzata evasiva nella direzione del primo sensore che ha rilevato l'ostacolo, senza interrompere la marcia.
- Dato che il primo sensore che ha rilevato l'ostacolo smette di rilevarlo mentre il secondo lo rileva ancora, quando questa condizione viene verificata, allora la sterzata evasiva viene attivata.
- Dato un ostacolo rilevato a distanza minore o uguale a 70 cm, quando questa condizione viene verificata, allora il rover comanda l'arresto indipendentemente dallo stato della manovra evasiva.

R19. Alta affidabilità livello 2

Requisito: La supervisione e il controllo devono essere realizzati in configurazione ad alta affidabilità con ridondanza e diversità a livello 2, mediante due board diverse che eseguono algoritmi distinti e condividono le informazioni di stato e fault necessarie alla decisione.

User Story: come progettista di affidabilità, voglio che due board indipendenti eseguano supervisione e controllo e condividano le informazioni di stato e fault, così da garantire il mantenimento di un comportamento sicuro anche in presenza di guasti o perdita di comunicazione.

Acceptance Criteria:

- Dato il sistema completo, quando si analizza l'architettura, allora risultano presenti due board diverse che eseguono funzioni di supervisione

e controllo in modo ridondante e con logiche autonome.

- Dato che le due board sono operative, quando viene rilevato un fault o un cambiamento di stato, allora tale informazione viene condivisa tra le board per consentire una decisione coerente.
- Dato un fault o una perdita di comunicazione su una board, quando l'altra board resta operativa, allora il sistema mantiene una supervisione coerente e porta il rover in uno stato sicuro o degradato secondo le politiche definite.

R20. Affidabilità nel trasferimento informazione

Requisito: Il sistema deve includere controllo (ed eventualmente correzione) dei trasferimenti informativi tra board.

User Story: come progettista comunicazione, voglio un protocollo robusto con validazione, riallineamento e controllo di freschezza, così da mantenere coerente lo stato condiviso anche con disturbi o perdita di fase.

Acceptance Criteria:

- Dato un dato ricevuto tra board, quando non supera i controlli di validità del protocollo, allora viene scartato e non usato per il controllo.
- Dato un disallineamento tra trasmettitore e ricevitore (fuori fase), quando viene rilevato, allora il canale RX deve riallinearsi automaticamente per riprendere l'acquisizione corretta dei messaggi.
- Dato che i dati scambiati includono indicatori di freschezza, quando l'aggiornamento non rispetta la finestra temporale attesa, allora il supervisore applica la politica di fault (degradato o critico) definita.

R21. Modalità degradata con singola board attuatrice

Requisito: In caso di malfunzionamento di una delle due board, il rover deve entrare in modalità degradata in cui una sola board mantiene il controllo dei motori, la board guasta viene isolata e viene applicato un limite di velocità per garantire la sicurezza.

User Story: come supervisore del sistema, voglio garantire continuità minima di controllo anche in presenza di fault di una board, così da mantenere

il rover in uno stato sicuro e controllato.

Acceptance Criteria:

- Dato un fault confermato su una delle due board, quando il supervisore rileva la condizione, allora il sistema entra in modalità degradata e la board guasta viene isolata.
- Dato che il sistema opera in modalità degradata, quando i motori sono controllati, allora l'attuazione è effettuata da una sola board operativa.
- Dato che il rover è in modalità degradata, quando vengono impartiti comandi di movimento, allora la velocità è limitata al valore massimo previsto per tale modalità.
- Dato che il rover è in modalità degradata, quando viene rilevato un ostacolo in qualsiasi direzione, allora il sistema comanda direttamente lo stop di emergenza.

R22. Protezione temperatura e batteria

Requisito: Il rover deve applicare le stesse limitazioni della modalità degradata quando temperatura o livello di batteria raggiungono le soglie di allarme, e arrestarsi al raggiungimento delle soglie critiche.

User Story: come supervisore del sistema, voglio prevenire condizioni pericolose dovute a temperatura elevata o batteria scarica, così da proteggere i componenti e mantenere la sicurezza operativa.

Acceptance Criteria:

- Dato che temperatura o livello di batteria raggiungono la soglia di allarme, quando il supervisore aggiorna lo stato, allora il sistema entra in modalità degradata applicando lo stesso limite di velocità definito per i fault di board.
- Dato che temperatura o livello di batteria raggiungono la soglia critica, quando la condizione viene confermata, allora il sistema arresta il rover e lo porta in uno stato sicuro.
- Le soglie di allarme e critiche sono definite con margine rispetto ai limiti fisici dei componenti, per garantire l'attivazione preventiva delle protezioni.

3.2 Requisiti su malfunzionamenti e soglie di allarme

M1. Una board in failure

Condizione di malfunzionamento: Una delle due board è in failure.

User Story: come progettista del supervisore, voglio gestire il fault di una sola board, così da mantenere un controllo sicuro in modalità degradata.

Acceptance Criteria:

- Dato fault confermato su una board, quando il supervisore aggiorna lo stato, allora il sistema passa in modalità degradata secondo R21.
- Dato ingresso in degradata per fault di board, quando il rover continua a operare, allora vengono applicati i limiti di sicurezza previsti.

M2. Entrambe le board in failure

Condizione di malfunzionamento: Tutte e due le board sono in failure.

User Story: come responsabile sicurezza, voglio che il rover si arresti quando falliscono entrambe le board, così da evitare perdita totale di controllo.

Acceptance Criteria:

- Dato fault confermato su entrambe le board, quando il supervisore valuta lo stato globale, allora viene imposto stato critico con stop di emergenza.
- Dato stato critico per failure doppia, quando il rover è fermo, allora resta in stato sicuro fino a reset/ripristino.

M3. Un motor controller non risponde

Condizione di malfunzionamento: Uno dei due motor controller non risponde ai comandi.

User Story: come responsabile sicurezza, voglio che un guasto di attuazione venga trattato come fault critico, così da portare subito il rover in stato sicuro.

Acceptance Criteria:

- Dato mancato riscontro di un motor controller, quando il fault è confermato, allora il supervisore imposta stato critico.
- Dato stato critico per guasto di attuazione, quando il sistema reagisce, allora il rover esegue stop di emergenza e segnala il fault.

M4. Entrambi i motor controller non rispondono

Condizione di malfunzionamento: Entrambi i motor controller non rispondono ai comandi.

User Story: come responsabile sicurezza, voglio che un guasto di attuazione venga trattato come fault critico, così da portare subito il rover in stato sicuro.

Acceptance Criteria:

- Dato mancato riscontro di entrambi i motor controller, quando il fault è confermato, allora il supervisore imposta stato critico.
- Dato stato critico per guasto di attuazione, quando il sistema reagisce, allora il rover esegue stop di emergenza e segnala il fault.

M5. Guasto encoder

Condizione di malfunzionamento: Uno (almeno) degli encoder non funziona.

User Story: come progettista del controllo, voglio gestire il guasto encoder con fallback sicuro, così da evitare instabilità di controllo.

Acceptance Criteria:

- Dato guasto confermato di almeno un encoder, quando il supervisore aggiorna lo stato, allora il sistema applica la strategia prevista (fallback degradato o stop).
- Dato fallback attivo, quando l'attuazione prosegue, allora vengono rispettati i limiti di sicurezza della modalità degradata.

M6. Guasto accelerometro

Condizione di malfunzionamento: L'accelerometro non funziona.

User Story: come progettista del supervisore, voglio classificare la perdita dati accelerometro in modo progressivo, così da distinguere condizioni degradate da condizioni critiche.

Acceptance Criteria:

- Dato ricezione accelerometro ridotta ma non assente, quando la condizione supera la soglia prevista, allora il supervisore imposta stato degradato.
- Dato timeout accelerometro (assenza totale dati), quando il fault è confermato, allora il supervisore imposta stato critico con stop di emergenza.

M7. Sistema non accetta comandi

Condizione di malfunzionamento: Il sistema non accetta comandi.

User Story: come operatore, voglio che la perdita dei comandi porti il rover a fermarsi in sicurezza, così da evitare movimenti non controllati.

Acceptance Criteria:

- Dato assenza o invalidità persistente dei comandi in ingresso, quando la condizione è confermata, allora viene generato fault comunicazione/input.
- Dato fault comunicazione/input attivo, quando il supervisore aggiorna lo stato, allora il rover passa a stato critico con arresto sicuro.

M8. Disconnessione controller remoto

Condizione di malfunzionamento: Il controller remoto si disconnette durante l'uso.

User Story: come operatore, voglio che una disconnessione improvvisa del controller porti il rover a comando neutro, così da ottenere un arresto ordinato e sicuro.

Acceptance Criteria:

- Dato perdita della connessione controller, quando il sistema rileva l'evento, allora i riferimenti comando vengono riportati a stato neutro.
- Dato comandi in stato neutro per disconnessione, quando il rover è in marcia, allora il sistema applica una fermata regolare fino a velocità nulla.

M9. Motori non rispondono ai comandi

Condizione di malfunzionamento: I motori non rispondono ai comandi.

User Story: come responsabile sicurezza, voglio che un guasto di attuazione venga trattato come fault critico, così da portare subito il rover in stato sicuro.

Acceptance Criteria:

- Dato mancato riscontro dei motori ai comandi inviati, quando il fault è confermato, allora il supervisore imposta stato critico.
- Dato accelerometro ed encoder disponibili, quando la risposta dei motori risulta incoerente con i comandi, allora il supervisore usa il controllo incrociato per rafforzare la diagnosi del guasto di attuazione.
- Dato stato critico per guasto di attuazione, quando il sistema reagisce, allora il rover esegue stop di emergenza e segnala il fault.

M10. Allarme temperatura

Condizione di malfunzionamento: Temperatura interna sopra soglia "DA DEFINIRE" per intervallo "DA DEFINIRE".

User Story: come progettista del supervisore, voglio gestire progressivamente l'aumento di temperatura, così da evitare danni termici ai componenti.

Acceptance Criteria:

- Dato temperatura oltre soglia di allarme per il tempo definito, quando il supervisore conferma l'evento, allora viene applicata limitazione di velocità (degradata) o stato critico secondo livello soglia.

- Dato evento termico confermato, quando il sistema aggiorna la diagnostica, allora l'allarme viene registrato nei log.

M11. Batteria sotto 23%

Condizione di malfunzionamento: La batteria scende sotto al 23% di carica.

User Story: come progettista del supervisore, voglio reagire a batteria bassa con politiche conservative, così da proteggere la batteria e mantenere sicurezza operativa.

Acceptance Criteria:

- Dato livello batteria in avvicinamento alla soglia del 23%, quando la condizione supera la soglia di preallarme, allora il supervisore imposta stato degradato.
- Dato livello batteria sotto al 23%, quando la condizione è confermata, allora il supervisore imposta stato critico con arresto del rover.
- Dato evento di batteria bassa, quando il supervisore aggiorna la diagnostica, allora la condizione viene tracciata nei log.

3.3 Requisiti di modellazione e progettazione del controllo

D1. Definizione stato globale

Requisito: Definire una rappresentazione dello stato globale del sistema.

User Story: come progettista del sistema, voglio rappresentare lo stato globale in forma sintetica e condivisibile, così da coordinare le decisioni tra le due board.

Acceptance Criteria:

- Dato il modello di supervisione, quando viene definito lo stato globale, allora include almeno classe di severità fault e validità delle informazioni condivise.

- Dato lo stato globale definito, quando viene usato per il controllo, allora evita dipendenza da dati ridondanti o non necessari alla decisione operativa.

D2. Ricostruzione stato globale

Requisito: Definire le fonti dati usate per ricostruire lo stato globale.

User Story: come progettista del sistema, voglio tracciare le sorgenti informative minime necessarie, così da costruire lo stato globale con soli dati utili.

Acceptance Criteria:

- Dato il set di segnali di progetto, quando si selezionano le fonti per stato globale, allora vengono incluse solo le informazioni necessarie a supervisione e attuazione.
- Dato ogni fonte selezionata, quando si verifica la tracciabilità, allora esiste un legame esplicito tra dato acquisito e decisione che abilita.

D3. Partizioni delle variabili di stato

Requisito: Definire due partizioni delle variabili di stato acquisite direttamente da ciascuna board.

User Story: come architetto del controllo distribuito, voglio partizionare le variabili tra le board in base all'osservabilità locale, così da avere responsabilità chiare.

Acceptance Criteria:

- Dato il modello a due board, quando si definiscono le partizioni, allora ogni variabile appartiene alla board che la misura o la produce direttamente.
- Dato la partizione definita, quando una board necessita di una variabile remota, allora tale variabile è acquisita tramite scambio esplicito.

D4. Protocollo scambio messaggi

Requisito: Definire protocollo di scambio informazioni tra partizioni per ricostruire stato globale.

User Story: come progettista comunicazione, voglio uno scambio robusto con controllo di validità e freschezza, così da evitare decisioni su dati stantii.

Acceptance Criteria:

- Dato lo scambio tra board, quando un dato remoto arriva, allora viene validato prima di essere usato nella ricostruzione dello stato globale.
- Dato un dato condiviso, quando manca evidenza di freschezza entro la finestra prevista, allora il dato viene considerato degradato o invalido secondo policy.
- Dato il canale di supervisione, quando si verifica la coerenza periodica, allora ogni board puo' confermare la vitalità logica dell'altra tramite indicatori dedicati.

D5. Automa stateflow per stati interni e globale

Requisito: Definire automa Stateflow per task, stati interni e stato globale.

User Story: come progettista del comportamento, voglio modellare in Stateflow lo stato operativo del rover e la logica di supervisione, così da avere decisioni coerenti tra guida, safety e fault handling.

Acceptance Criteria:

- Dato il modello Stateflow, quando si analizzano gli stati operativi del rover, allora sono presenti gli stati necessari alle manovre nominali e alle manovre di sicurezza.
- Dato i contributi di stato locale delle due board, quando il supervisore aggiorna lo stato globale, allora lo stato globale riflette in modo deterministico la combinazione delle due viste locali.
- Dato una condizione anomala rilevata da logiche di confronto tra segnali disponibili, quando la condizione è confermata, allora il risultato viene propagato alla decisione di stato del rover.

D7. Ciclo acquisizione-elaborazione-attuazione

Requisito: Modellare il ciclo acquisizione-elaborazione-attuazione in funzione dello stato corrente e dello stato obiettivo.

User Story: come progettista controllo, voglio che il ciclo acquisizione-elaborazione-attuazione usi come stato corrente lo stato globale del rover e come stato obiettivo il comando utente da joystick, così da avere riferimenti coerenti con la situazione reale.

Acceptance Criteria:

- Dato stato globale aggiornato del rover e comando joystick valido, quando il ciclo viene eseguito, allora i riferimenti di attuazione risultano coerenti sia con l'obiettivo utente sia con i vincoli di sicurezza.
- Dato variazione dello stato globale del rover, quando il ciclo viene aggiornato, allora i riferimenti vengono ricalcolati in modo coerente con il nuovo stato operativo.
- Dato comando utente valido, quando i sonar rilevano una condizione di sicurezza, allora il riferimento di attuazione viene corretto secondo la logica di sterzata/schivata prevista.

D8. Definizione del comportamento di errore

Requisito: Definire error handlers (trasmissione, deadline, raggiungibilità stati critici, altri errori).

User Story: come progettista safety, voglio error handler esplicativi per timeout, comunicazione e superamento soglie operative, così da garantire reazioni coerenti tra stato degradato e stato critico.

Acceptance Criteria:

- Dato assenza dati o timeout su periferiche/comunicazione, quando la soglia di diagnosi è superata, allora l'handler imposta la classe di fault prevista e la relativa reazione.
- Dato degrado progressivo (es. dati meno freschi del previsto o avvicinamento a soglie operative), quando la condizione viene confermata, allora l'handler attiva modalità degradata.
- Dato violazione deadline o errore logico, quando l'handler corrispondente è attivato, allora il sistema applica la reazione di sicurezza prevista (degradata o critico con stop) e registra l'evento nei log.

D9. Definizione e gestione degli stati di funzionamento degradato e di stop di emergenza

Requisito: Definire, progettare e codificare le condizioni di ingresso e la gestione degli stati di funzionamento degradato e di stop di emergenza.

User Story: come progettista software, voglio definire e implementare in codice le condizioni di ingresso e la gestione degli stati di funzionamento degradato e di stop di emergenza, così da garantire un comportamento sicuro e coerente con la specifica del sistema.

Acceptance Criteria:

- Dato un insieme di condizioni critiche o di funzionamento degradato definite a specifica, quando tali condizioni si verificano, allora il sistema entra nello stato corrispondente.
- Dato lo stato di funzionamento degradato o di stop di emergenza, quando il sistema si trova in tale stato, allora il comportamento operativo risulta coerente con quanto previsto dalla specifica.
- Dato un evento che richiede la transizione verso uno stato degradato o di stop di emergenza, quando viene rilevato dal software, allora la transizione viene eseguita correttamente.

3.4 Requisiti di schedulazione

S1. Progettare Task real time per il ciclo attuazione-elaborazione-acquisizione

Requisito: Progettare insieme task real-time per acquisizione/elaborazione/attuazione e politica di scheduling.

User Story: come progettista real-time, voglio una schedulazione coerente tra acquisizione, elaborazione e attuazione, così da mantenere controllo stabile e tempi prevedibili.

Acceptance Criteria:

- Dato il set di task principali, quando viene definita la schedulazione, allora ogni task ha periodo e priorità compatibili con la funzione che svolge.

- Dato l'algoritmo di scheduling adottato, quando si verifica il set di task, allora risultano rispettati i bound di schedulabilità previsti.
- Dato una finestra temporale di osservazione, quando il sistema opera in nominale, allora acquisizione, elaborazione e attuazione rispettano l'ordine previsto senza perdita di coerenza.

S2. Progettare task per funzionamento in stato degradato e critico

Requisito: Progettare task per stato degradato e stop emergenza, incluse interazioni con task ordinari.

User Story: come responsabile safety, voglio che i task restino coerenti con la modalità operativa (nominale, degradata, critica) definita dal supervisore, così da avere una reazione uniforme del rover.

Acceptance Criteria:

- Dato ingresso in stato degradato, quando il supervisore aggiorna la modalità operativa, allora i task applicano riferimenti coerenti con i limiti di sicurezza attivi.
- Dato ingresso in stato critico, quando viene richiesto stop di emergenza, allora i task di attuazione ricevono riferimenti di fermo sicuro.
- Dato una board non attiva per l'attuazione, quando i task elaborano i riferimenti, allora non viene prodotto comando motore attivo verso l'hardware.

S3. Progettare le dipendenze tra precedenze di task

Requisito: Progettare un meccanismo per gestire dipendenze di precedenza tra task con scheduler noti.

User Story: come progettista software, voglio vincoli di precedenza espliciti tra task dipendenti e una politica di lettura coerente dei dati condivisi, così da mantenere continuità operativa.

Acceptance Criteria:

- Dato una catena funzionale tra task dipendenti, quando viene configurata la schedulazione, allora le precedenze sono definite in modo esplicito.
- Dato una dipendenza di precedenza, quando il task a monte non produce un nuovo campione, allora il task a valle usa l'ultimo dato prodotto e valido.

3.5 Parametri scelti in fase di progettazione

Nel seguito sono riportati i parametri fissati durante la fase di progettazione, richiesti dalle specifiche di progetto.

Soglie Temperatura Le soglie di temperatura sono state ricavate in modo conservativo considerando tutti i componenti presenti sul rover e i relativi limiti di funzionamento da datasheet. Come riferimento di sistema è stata usata l'intersezione dei range ammissibili, ottenendo una finestra nominale circa $[-20, +70]^\circ\text{C}$.

Per la diagnostica operativa è stato introdotto un margine rispetto ai limiti nominali: soglia critica a temperatura maggiore di 65°C o minore di -15°C , e soglia degradata a temperatura maggiore di 55°C o minore di -5°C . Per confermare la condizione termica si richiede permanenza oltre soglia per almeno 4 secondi, così da ridurre falsi allarmi dovuti a picchi rapidi o a imprecisione della misura ADC.

Finestra temporale per ostacolo In movimento La logica di gestione dell'ostacolo in movimento viene applicata nella fascia operativa dei sonar compresa tra circa 1.5 m e 3 m. Oltre i 3 m la presenza dell'ostacolo non è garantita, mentre sotto circa 0.7 m è previsto uno stop di emergenza. La fascia 1.5–3 m rappresenta quindi l'intervallo in cui viene presa la decisione di schivata.

La logica è definita tra coppie di sonar adiacenti (sinistro-centrale oppure centrale-destro). Quando uno dei due sonar della coppia (indicato come S_1) smette di rilevare un ostacolo precedentemente presente, il sistema entra in uno stato di attesa. Se il sonar adiacente (S_2) rileva immediatamente la presenza, viene avviata la manovra di schivata. In caso contrario, il sistema attende per un tempo massimo pari a $T_{win} = 3 \text{ s}$. Se durante questo intervallo

il sonar adiacente non rileva alcun ostacolo, il sistema considera concluso l'evento senza eseguire manovre, assumendo che la perdita di rilevazione fosse dovuta a un transitorio o a un ostacolo non rilevante per la traiettoria. La scelta di $T_{win} = 3$ s è basata sulla fascia operativa dei sonar compresa tra 1.5 m e 3 m, ovvero l'intervallo in cui viene presa la decisione di schivata. La larghezza di tale fascia è pari a

$$\Delta d = 3 - 1.5 = 1.5 \text{ m.}$$

Assumendo una velocità di riferimento pari a circa metà della velocità massima del rover,

$$v_{\text{ref}} \approx 0.65 \text{ m/s},$$

il tempo caratteristico associato all'attraversamento di questa fascia risulta:

$$t = \frac{1.5 \text{ m}}{0.65 \text{ m/s}} \approx 2.3 \text{ s.}$$

La scelta di una finestra temporale pari a 3 s risulta quindi coerente con questo tempo caratteristico e introduce un margine di sicurezza per compensare eventuali ritardi di rilevazione o perdite temporanee di misura. In questo modo la finestra consente di distinguere in modo robusto tra la continuità della stessa rilevazione e la comparsa di un nuovo ostacolo.

Limite Velocità in Modalità Degradata In modalità degradata viene applicata uno scaling del riferimento pari a 0.5, questo dimezza il comando di velocità e, di conseguenza, riduce implicitamente la velocità massima effettiva del rover alla metà di quella nominale. Il valore 0.5 è stato scelto come compromesso tra sicurezza e usabilità. Da un lato il rover resta ancora comandabile con buona reattività e senza diventare eccessivamente lento, dall'altro la riduzione di velocità evita manovre troppo aggressive in condizioni di funzionamento degradato del rover.

Questa scelta è coerente con le condizioni che determinano l'ingresso in modalità degradata: come la riduzione della frequenza dei dati validi disponibili oppure avvicinamento a soglie di temperatura o batteria. Operando a velocità ridotta, la dinamica del rover risulta meno aggressiva e il tempo disponibile per la reazione aumenta, migliorando la robustezza complessiva del sistema rispetto alla modalità nominale.

Combo Retromarcia/180 La commutazione tra retromarcia normale e modalità 180 è definita tramite la sequenza comandi utente `btn1 btn1 btn2`

btn2. Tra un evento di pressione e il successivo non deve trascorrere piu' di 1 s. Il riconoscimento avviene sul fronte di rilascio (falling) del pulsante, cosi da evitare ripetizioni involontarie dovute al mantenimento in pressione.

4 Funzionamento generale del software di ciascuna board

4.1 Interazione tra Task e Meccanismo di Snapshot

Ciascuna board gestisce sette task, ognuno dei quali attinge, secondo necessità, alle informazioni richieste dalla propria logica di controllo tramite gli snapshot, ovvero variabili globali condivise. Si consideri, ad esempio, la Figura 2 che illustra il funzionamento dei task *SupervisorB1* e *Transmit* che vengono eseguiti sulla Board 1. Al termine della propria esecuzione, ogni task può aggiornare tali snapshot, rendendo i nuovi dati disponibili agli altri task. Nell'esempio, il task *SupervisorB1* aggiorna lo snapshot **supervisor_snapshot**, che viene poi utilizzato dal task *Transmit* per l'invio dei dati verso la Board2. Allo stesso modo, la Board 2 tiene legge i scrive i propri snapshot, come illustrato nella Figura 3.

Ogni snapshot integra, oltre ai dati specifici, due variabili di monitoraggio temporale:

- *data_last_valid_ms*: indica l'istante temporale dell'ultimo aggiornamento dei dati ritenuto valido (ad esempio, l'ultima lettura coerente ricevuta dagli encoder).
- *task_last_run_ms*: riporta l'ultimo istante in cui il task è stato effettivamente eseguito.

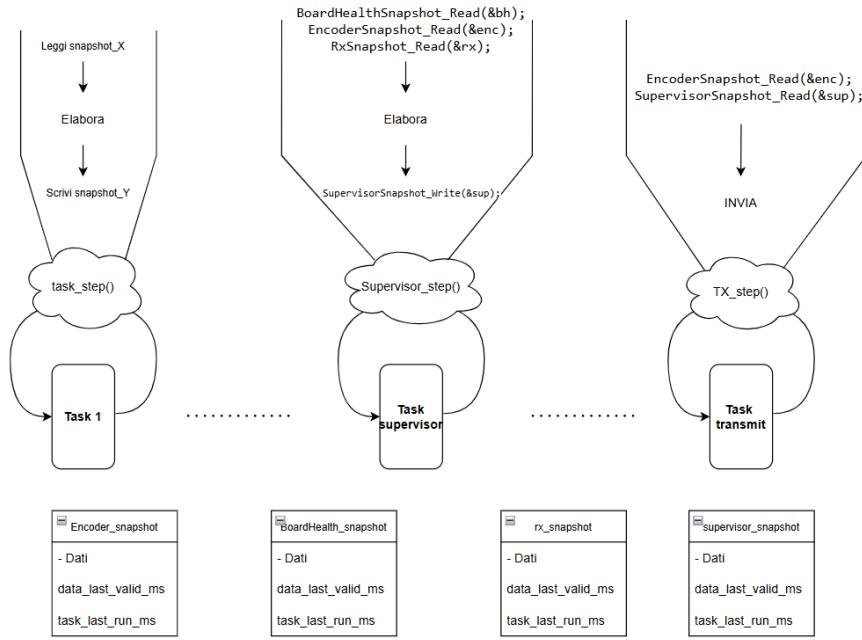


Figure 2: Funzionamento generale del software di Board 1.

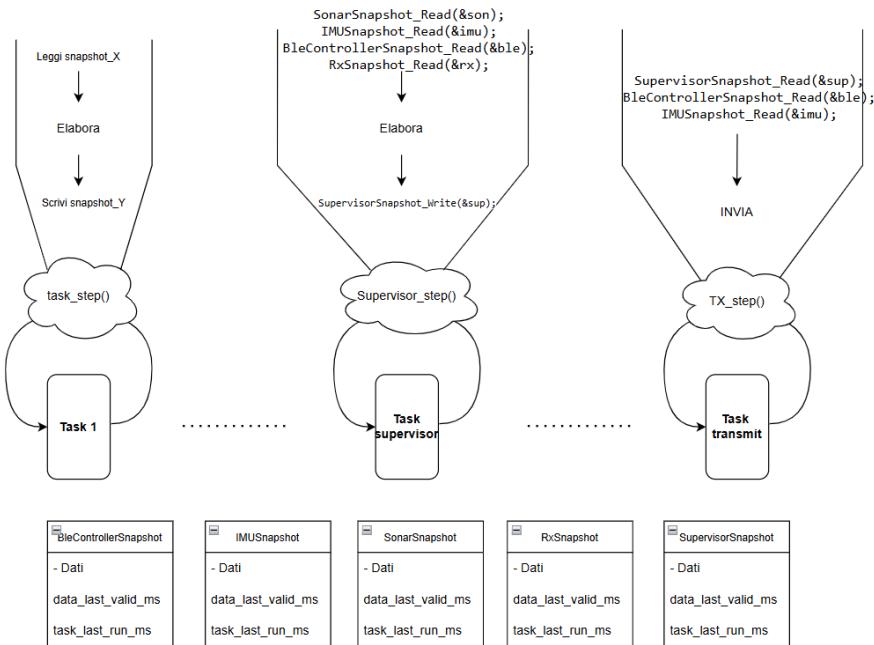


Figure 3: Funzionamento generale del software di Board 2.

4.2 Funzionamento task di Board 2

4.2.1 Task: acquisizione distanza dagli ostacoli

La Board 2 è equipaggiata con tre sensori ad ultrasuoni **HC-SR04** per il rilevamento di ostacoli. Questi sono disposti a 45° l'uno dall'altro, come mostrato in Figura 4.

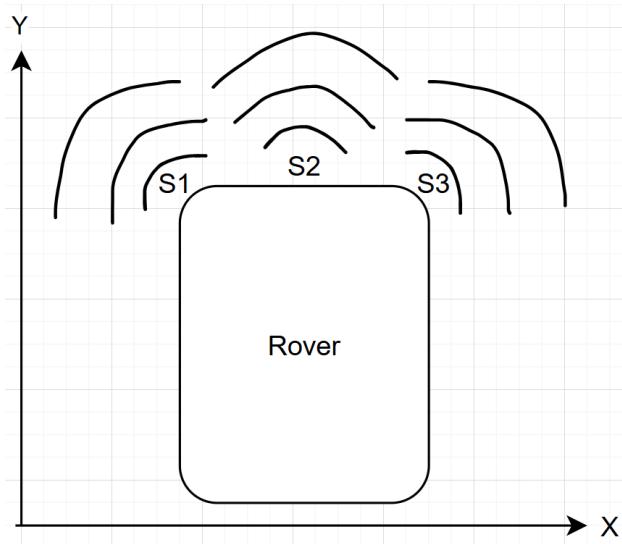


Figure 4: Disposizione dei sensori sulla Board 2.

Ogni sensore emette onde sonore ad alta frequenza e produce segnali di tipo onda quadra la cui durata è proporzionale all'ostacolo rilevato.



Figure 5: Segnale generato dal sensore HC-SR04 in presenza di un ostacolo a 3 metri di distanza.

La board2, rilevando i fronti di salita e discesa, può misurare l'intervallo tra i due fronti e utilizzare questa informazione per calcolare la distanza dall'ostacolo e prendere decisioni appropriate per evitare collisioni.

Utilizzo DMA per la lettura dei segnali Per ottimizzare la lettura dei segnali dai sensori ad ultrasuoni, la Board 2 utilizza il Direct Memory Access (DMA). Il DMA consente di trasferire i dati direttamente tra la periferica (i sensori ad ultrasuoni) e la memoria, senza l'intervento della CPU. Il timer utilizzato è il *Timer1*, con i canali 1, 2 e 3 configurati in modalità *input capture* per catturare i fronti di salita e discesa generati dai tre sensori.

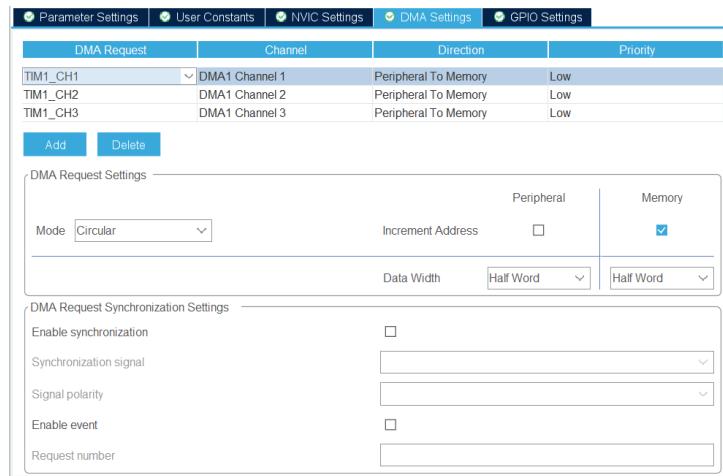


Figure 6: Configurazione del DMA per la lettura dei segnali dai sensori.

Ogni canale del DMA è configurato in modalità interrupt, permettendo, alla fine della rilevazione dei due fronti (salita e discesa), di eseguire una *Callback* che imposta dei flag a 1. Questo flag indica che i fronti sono stati rilevati e che la distanza dall'ostacolo può essere calcolata. In totale vengono eseguite solo 3 callback, attivate solo quando uno specifico canale DMA ha terminato la lettura di entrambi i fronti. La Figura 7 mostra un esempio di callback eseguita al termine della rilevazione dei fronti.

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    if(htim->Instance == TIM1){
        switch(htim->Channel){
            case HAL_TIM_ACTIVE_CHANNEL_1:
                if(flag.sonar1_ok == 0){
                    flag.sonar1_ok = 1;
                    sonar_count++;
                }
                break;
            case HAL_TIM_ACTIVE_CHANNEL_2:
                if(flag.sonar2_ok == 0){
                    flag.sonar2_ok = 1;
                    sonar_count++;
                }
                break;
            case HAL_TIM_ACTIVE_CHANNEL_3:
                if(flag.sonar3_ok == 0){
                    flag.sonar3_ok = 1;
                    sonar_count++;
                }
                break;
            default:
                break;
        }
    }

    if (sonar_count >= 3) {
        // Notifica il task e richiedi uno switch immediato se necessario
        xTaskNotifyFromISR(sonarTaskHandle, 0, eNoAction, &xHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
    }
}

```

Figure 7: Callback eseguita al termine della rilevazione dei fronti.

Alla fine della lettura, il task aggiorna lo snapshot **SonarsSnapshot** con le distanze rilevate dai tre sensori, come mostrato in Figura 8.

```

typedef struct
{
    uint16_t dist_cm[3];

    uint32_t task_last_run_ms;      /* ultima esecuzione del task */
    uint32_t data_last_valid_ms[3]; /* ultimo istante in cui i dati sono validi */
} SonarSnapshot_t;

```

Figure 8: Struttura dati dello snapshot SonarsSnapshot.

4.2.2 Task: lettura comandi utente

La Board 2 riceve i comandi utente provenienti da un joystick che comunica con un'ESP32 tramite Bluetooth. La Board 2, attraverso un task dedicato, riceve i comandi provenienti dalla *ESP32* con il protocollo **I2Ce** li integra nello snapshot **BleControllerSnapshot**. In Figura 9 è mostrata lo step del task dedicato alla ricezione dei comandi utente, in cui viene eseguita la lettura dei dati provenienti dalla *ESP32* e l'aggiornamento dello snapshot **BleControllerSnapshot**.

```

void BleController_TaskStep(void)
{
    static BleControllerSnapshot_t snap;
    BleRawFrame_t frame;

    uint32_t now = osKernelGetTickCount();
    snap.task_last_run_ms = now;

    BleI2CStatus_t st = BleController_I2C_ReadFrame(&frame);

    if (st == BLE_I2C_COMPLETE)
    {
        snap.data_last_valid_ms = now;

        /* Normalizzazione assi joystick A */
        snap.ax_norm = NormalizeAxis(frame.ax);
        snap.ay_norm = NormalizeAxis(frame.ay);

        snap.bx_norm = NormalizeAxis(frame.bx);
        snap.by_norm = NormalizeAxis(frame.by);

        /* Pulsanti */
        snap.a_btn = frame.a_btn;
        snap.b_btn = frame.b_btn;
        snap.btn1 = frame.btn1;
        snap.btn2 = frame.btn2;
    }

    BleControllerSnapshot_Write(&snap);
}

```

Figure 9: Step del task dedicato alla ricezione dei comandi utente.

La funzione *BleController_I2C_ReadFrame* si occupa di leggere i dati provenienti dalla *ESP32* e di restituirli in una struttura dati, che viene poi utilizzata per aggiornare lo snapshot **BleControllerSnapshot**.

In Figura 10 è mostrata la struttura dati dello snapshot **BleControllerSnapshot**, in cui sono presenti i comandi utente normalizzati (*x_norm* e *y_norm*).

```

typedef struct
{
    uint32_t task_last_run_ms;      /* ultima esecuzione del task */
    uint32_t data_last_valid_ms;   /* ultimo istante in cui i dati sono validi */

    /* Stick A normalizzato */
    float ax_norm;
    float ay_norm;

    /* Stick B normalizzato */
    float bx_norm;
    float by_norm;

    /* Pulsanti */
    uint8_t a_btn;
    uint8_t b_btn;
    uint8_t btn1;
    uint8_t btn2;

} BleControllerSnapshot_t;

```

Figure 10: Struttura dati dello snapshot BleControllerSnapshot.

Interfacciamento DualSense PS5 tramite ESP32 Per la generazione dei comandi remoti è stato utilizzato un controller **Sony DualSense (PS5)**, interfacciato a una scheda **ESP32** che funge da bridge tra il protocollo Bluetooth e il bus I2C del rover. L’implementazione sul lato ESP32 è stata realizzata in ambiente Arduino IDE sfruttando la libreria **Bluepad32**. I dati del joystick sono stati rappresentati tramite la struct presente nell’immagine 12.



Figure 11: Tasti presenti sul joystick utilizzato.

```

// 1. DEFINIZIONE STRUTTURA (Identica alla tua originale)
// L'attributo packed assicura che non ci siano spazi vuoti tra i dati (totale 12 byte)
struct __attribute__((packed)) controller_t {
    uint16_t ax;      // Joystick Sinistro X
    uint16_t ay;      // Joystick Sinistro Y
    uint8_t a_btn;    // Pressione L3
    uint16_t bx;      // Joystick Destro X
    uint16_t by;      // Joystick Destro Y
    uint8_t b_btn;    // Pressione R3
    uint8_t btn1;     // Tasto Croce
    uint8_t btn2;     // Tasto Cerchio
};

// Valori di riposo (Standby)
controller_t controller_standby = {255, 255, 0, 255, 255, 0, 0, 0};
controller_t controller_data;

```

Figure 12: Rappresentazione tasti del controller

Una volta rappresentati i dati e individuato i valori di standby, il flusso di elaborazione sull’ESP32 segue i seguenti passaggi:

1. **Pairing e Connessione:** All’avvio, l’ESP32 scansiona i dispositivi Bluetooth. Una volta messo il DualSense in modalità pairing, la libreria stabilisce una connessione sicura.
2. **Parsing dei dati:** Tramite le callback della libreria (`onConnectedGamepad`), vengono eseguiti alcuni controlli per garantire che si possa connettere solo il joystick autorizzato alla scheda.

```

// 2. CALLBACK: COSA SUCCIDE QUANDO IL JOYSTICK SI CONNETTE
void onConnectedGamepad(GamepadPtr gp) {
    const uint8_t* addr = gp->getProperties().btaddr;

    // Controllo ID Sony e ID DualSense
    bool isPS5 = (gp->getProperties().vendor_id == 0x054c &&
                  (gp->getProperties().product_id == 0x0ce6 || gp->getProperties().product_id == 0x0df2));

    if (isPS5) {
        Serial.println("Rilevato: Controller PS5 (DualSense) ORIGINALE.");
    } else {
        Serial.printf("Rilevato controller non PS5 (VID: %04x, PID: %04x). Rifiuto...\n",
                     gp->getProperties().vendor_id, gp->getProperties().product_id);
        gp->disconnect();
        return;
    }
    Serial.printf("Tentativo di connessione da: %02X:%02X:%02X:%02X:%02X:%02X\n",
                 addr[0], addr[1], addr[2], addr[3], addr[4], addr[5]);

    // Confronto del MAC
    if (memcmp(addr, authorizedMac, 6) == 0 && myGamepads[0] == nullptr) {
        Serial.println("DualSense AUTORIZZATO!");
        gp->setColorLED(255, 0, 0); // Rosso per segnalare che la connessione è avvenuta

        // Lo mettiamo sempre nella posizione 0 per semplicità
        myGamepads[0] = gp;
    } else {
        Serial.println("ERRORE: Controller NON autorizzato. Rifiuto connessione...");
        gp->disconnect(); // Segnala allo stack di chiudere
    }
}

```

Figure 13: Implementazione della funzione di callback per l’inizializzazione del controller DualSense all’atto della connessione Bluetooth.

3. **Normalizzazione:** Viene continuamente monitorato lo stato del BLUETOOTH e, se il joystick è connesso, i valori provenienti da esso vengono mappati in un range normalizzato a seconda delle necessità del supervisore. Una volta mappati, tali valori vengono utilizzati per aggiornare i dati presenti in controller_data.

```

void loop() {
    // Aggiorna lo stato del Bluetooth (obbligatorio)
    BP32.update();

    // Usiamo solo lo slot 0 che abbiamo popolato SOLO se il MAC era corretto
    GamepadPtr gp = myGamepads[0];
    if (gp && gp->isConnected()) {
        const uint8_t* addr = gp->getProperties().btaddr;

        controller_data.ax = map(gp->axisX(), -511, 511, 511, 0);
        controller_data.ay = map(gp->axisY(), -511, 511, 511, 0);
        controller_data.bx = map(gp->axisRX(), -511, 511, 511, 0);
        controller_data.by = map(gp->axisRY(), -511, 511, 511, 0);

        uint16_t buttons = gp->buttons();
        controller_data.a_btn = (buttons & 0x0100) ? 1 : 0;
        controller_data.b_btn = (buttons & 0x0200) ? 1 : 0;
        controller_data.btn1 = (buttons & 0x0001) ? 1 : 0;
        controller_data.btn2 = (buttons & 0x0002) ? 1 : 0;
    }
    // Piccolo delay per non saturare la CPU
    delay(1);
}

```

Figure 14: Normalizzazione dei dati inviati alla ESP32 tramite joystick

4. **Trasmissione I2C:** L'ESP32 è configurata come **I2C Slave**. Quando la Board 2 interroga l'ESP32 (tramite la funzione BleController_I2C_ReadFrame citata precedentemente), l'ESP32 trasmette un pacchetto dati strutturato contenente lo stato dei pulsanti e la posizione degli analogici.

```

// 4. RISPOSTA I2C: QUANDO IL MASTER CHIEDE DATI
void onRequest() {
    // Invia i 12 byte della struttura tramite il bus I2C
    Wire.write((byte *)&controller_data, sizeof(controller_t));
}

```

Figure 15: Trasmissione dei dati del joystick della ESP32 al Master tramite callback

5. **Disconnessione:** Nel momento in cui viene rilevata la disconnessione del joystick dalla scheda, vengono impostati, tramite callback, i dati di default, che indicano al rover di rimanere fermo(velocità lineare e angolare impostate a zero). Così facendo, se il joystick dovesse disconnettersi per qualche motivo il rover rimane fermo e non vengono svolte azioni indesiderate che possono portarlo a situazioni pericolose.

```

// 3. CALLBACK: COSA SUCCIDE QUANDO IL JOYSTICK SI SCOLLEGA
void onDisconnectedGamepad(GamepadPtr gp) {
    if (myGamepads[0] == gp) {
        myGamepads[0] = nullptr;
        controller_data = controller_standby;
        Serial.println("DualSense autorizzato disconnesso.");
    }
}

```

Figure 16: Normalizzazione dei dati inviati alla ESP32 tramite joystick

Questa architettura permette di delegare il carico computazionale della gestione Bluetooth alla ESP32, garantendo che la Board 2 (STM32) rimanga focalizzata esclusivamente sui task real-time di supervisione e sicurezza.

4.2.3 Task: lettura giroscopio

La Board 2 è equipaggiata con un sensore IMU **MPU-6050**, che integra un accelerometro e un giroscopio a 3 assi. Poiché il sensore non fornisce direttamente l'orientamento assoluto, l'angolo di yaw (la rotazione del rover attorno al suo asse verticale) viene calcolato nel firmware integrando nel tempo i dati della velocità angolare provenienti dal giroscopio. All'accensione, il sistema esegue una calibrazione per impostare lo zero relativo alla direzione di avvio.

La comunicazione con l'IMU avviene tramite protocollo I2C, e i dati elaborati aggiornano ciclicamente lo snapshot IMUSnapshot (Figura 17).

```
typedef struct
{
    uint32_t task_last_run_ms;      /* ultima esecuzione del task */
    uint32_t data_last_valid_ms;   /* ultimo istante in cui i dati sono validi */

    /* Accelerometer (g) */
    float ax_g;
    float ay_g;
    float az_g;

    /* Gyroscope (deg/s) */
    float gx_dps;
    float gy_dps;
    float gz_dps;

    /* Temperature (°C) */
    float temperature_degC;

    float yaw;
} IMUSnapshot_t;
```

Figure 17: Struttura dati dello snapshot IMUSnapshot.

4.2.4 Task: supervisore Board 2

Il cuore logico della Board 2 è rappresentato dal task del supervisore, un processo che implementa una macchina a stati complessa generata tramite Simulink Stateflow. Il task opera come un arbitro di sicurezza tra i comandi utente e l'ambiente circostante, garantendo che il rover si muova nella direzione voluta dall'utente a meno di possibili ostacoli statici o in movimento. In Figura 18 è mostrato lo step del task del supervisore, in cui vengono evidenziati gli input e gli output.

```

void Supervisor_TaskStep(void)
{
    static SonarSnapshot_t son;
    static IMUSnapshot_t imu;
    static BleControllerSnapshot_t ble;
    static RxSnapshot_t rx;
    static SupervisorSnapshot_t sup;
    static uint8_t last_sup_counter = 0;
    static uint32_t last_sup_update_ms = 0;

    SonarSnapshot_Read(&son);
    IMUSnapshot_Read(&imu);
    BleControllerSnapshot_Read(&ble);
    RxSnapshot_Read(&rx);

    uint32_t now = osKernelGetTickCount();

    CommPayloadB1_t payload = rx.payload;

    if (payload.alive_counter != last_sup_counter) {
        last_sup_counter = payload.alive_counter;
        last_sup_update_ms = now;
    }

    SupervisorB2_U.Board1_Data= rx;
    SupervisorB2_U.BLE = ble;
    SupervisorB2_U.IMU = imu;
    SupervisorB2_U.Sonars = son;
    SupervisorB2_U.last_valid_b1_ms = last_sup_update_ms;
    SupervisorB2_U.now_ms = now;

    SupervisorB2_step();

    sup.critical_mask = SupervisorB2_Y.critical_mask;
    sup.degraded_mask = SupervisorB2_Y.degraded_mask;
    sup.command = SupervisorB2_Y.B2Decision;
    sup.isMotionConsistent = SupervisorB2_Y.isMotionConsistent;

    sup.alive_counter++;
    sup.task_last_run_ms = now;
}

```

Figure 18: Step del task del supervisore della Board 2.

Ad ogni ciclo, il task acquisisce i dati dai sensori locali (**Sonar**, **IMU**, **Comandi utente**) e le informazioni provenienti dalla Board 1 tramite lo snapshot **RxSnapshot**. Il sistema valuta l'integrità dei dati e la persistenza della comunicazione con la Board 1. In base alla latenza dei messaggi e alla coerenza dei sensori, il supervisore può far transitare il rover in stati di operatività *Degraded* o *Critical*. Utilizzando i dati dei tre sonar frontali, il supervisore implementa algoritmi di protezione, infatti, se viene rilevato un ostacolo a distanza critica (< 70 cm), il sistema forza un co-

mando di *CMD_ESTOP*. In caso di ostacoli in movimento (range 75 – 150 cm), il supervisore attiva delle procedure di deviazione (*CMD_GO_LEFT*, *CMD_GO_RIGHT*) o di aggiramento (*CMD_AVOID*), calcolando i nuovi target di angolo yaw necessari per direzionare il rover verso l'assenza di ostacoli.

Nel paragrafo "Coerenza di movimento del rover" si è parlato della capacità della Board 2 di monitorare la coerenza del movimento del rover. In effetti, il supervisore esegue un test di coerenza tra la rotazione stimata dalla lettura della velocità dei motori e quella misurata dall'IMU. Questa logica permette di rilevare anomalie meccaniche (come lo slittamento di una ruota o il bloccaggio di un motore) o anomalie dei sensori IMU ed encoder. Se ci sono incoerenze, il supervisore di Board 2 imposta un flag *isMotionConsistent* a 0.

Infine, in condizioni nominali, la Board 2 non invia comandi di attuazione ai motori, lasciando che sia la Board 1 ad attuare. Tuttavia, il supervisore consente alla Board 2 di attuare se si verifica una di queste condizioni:

- Viene rilevato un timeout critico nella ricezione dei dati dalla Board 1 (> 120 ms).
- Il supervisore della Board 1 funziona in maniera discontinua.
- Board 1 invia a Board 2 l'autorizzazione ad attuare.

E' da notare che l'attuazione passa attraverso il rele comandato da Board 1, quindi dovrà essere sempre Board 1 ad aprirlo per consentire al segnale UART di Board 2 di arrivare ai driver dei motori.

In Figura 19 è mostrata la *actuation_step* di Board 2 in funzione della variabile *authorized_to_send_command* in uscita dal supervisore.

```

bool authorized_to_send_command = SupervisorB2_Y.authorized_to_send_command;
if(authorized_to_send_command){

    static bool initialized = false;
    if(!initialized){
        Actuation_Init();
        initialized = true;
    }
    else{
        float v_ref = SupervisorB2_Y.v_ref_actuation;
        float omega_ref = SupervisorB2_Y.omega_ref_actuation;

        Actuation_Step(v_ref, omega_ref);

        bool emergency_stop_requested = SupervisorB2_Y.actuate_emergency_stop;
        if(emergency_stop_requested){
            HAL_GPIO_WritePin(ESTOP_GPIO_Port, ESTOP_Pin, GPIO_PIN_RESET);
        }
        else{
            HAL_GPIO_WritePin(ESTOP_GPIO_Port, ESTOP_Pin, GPIO_PIN_SET);
        }
    }
}

SupervisorSnapshot_Write(&sup);
}

```

Figure 19: Step di attuazione di Board 2 in funzione della variabile authorized_to_send_command.

Data l'assenza degli encoder, l'attuazione di Board 2 dovrà essere in open loop, basandosi esclusivamente sui comandi utente ricevuti e sui dati dei sensori, senza feedback diretto sulla velocità effettiva del rover. La funzione di attuazione trasforma i riferimenti di velocità del supervisore in segnali di potenza per i motori. Il sistema opera in anello aperto (open-loop), convertendo i giri al minuto (RPM) desiderati in tensione elettrica. Infine, i comandi vengono inviati al driver di potenza Sabertooth tramite una scalatura lineare del voltaggio in uscita, con un range di $[-6V; +6V]$ corrispondente a $[-MAX_RPM_DEGRADED; +MAX_RPM_DEGRADED]$.

Lo snapshot del supervisore di Board 2 è mostrato in Figura 20:

```

/* ===== Snapshot Supervisore Board 2 ===== */
typedef struct
{
    /* Timestamp decisione */
    uint32_t task_last_run_ms;

    /* Fault di board 2 presenti che comportano uno stato degradato*/
    uint32_t degraded_mask;

    /* Fault di board 2 presenti che comportano uno stato di emergenza*/
    uint32_t critical_mask;

    /* Comando semantico deciso da board 2 */
    SupervisorCommand_t command;

    /* Coerenza dei dati ricevuti da imu, utile a rilevare motor fault */
    bool isMotionConsistent;

    /* Heartbeat / alive counter */
    uint32_t alive_counter;
} SupervisorSnapshot_t;

```

Figure 20: Snapshot del supervisore di Board 2

4.2.5 Task: log dei dati per il debug

La Board 2 integra un task dedicato alla stampa dei dati per scopi di debug. Questo task, eseguito a cadenza regolare, acquisisce gli snapshot:

- RxSnapshot
- BleControllerSnapshot
- SonarsSnapshot
- IMUSnapshot

e stampa i dati più rilevanti su console. Questa funzionalità è fondamentale per monitorare lo stato del sistema durante le fasi di sviluppo e test, permettendo di identificare rapidamente eventuali anomalie o comportamenti imprevisti. Lo step del task è mostrato in Figura 21, in cui vengono evidenziati i dati acquisiti e stampati su console.

```

void Log_TaskStep(void)
{
    static char log_buf[LOG_BUF_LEN];

    BleControllerSnapshot_t ble;
    IMUSnapshot_t imu;
    SonarSnapshot_t sonar;
    RxSnapshot_t rx;

    BleControllerSnapshot_Read(&ble);
    IMUSnapshot_Read(&imu);
    SonarSnapshot_Read(&sonar);
    RxSnapshot_Read(&rx);

    Log_FormatSnapshot(log_buf, LOG_BUF_LEN,
                       &ble, &imu, &sonar, &rx);

    printf("%s", log_buf);
}

```

Figure 21: Step del task dedicato alla stampa dei dati per scopi di debug.

4.3 Funzionamento task di Board 1

4.3.1 Task: lettura batteria e temperatura

La Board 1 è equipaggiata con sensori per il monitoraggio della temperatura e della batteria. Un task dedicato esegue ciclicamente la lettura di questi sensori, aggiornando lo snapshot **BoardHealthSnapshot** con i valori rilevati.

In Figura 22 è mostrato lo step del task dedicato alla lettura della batteria e temperatura, in cui vengono evidenziati i dati acquisiti e aggiornati nello snapshot **BoardHealthSnapshot**.

```

void BoardHealth_TaskStep(void)
{
    static BoardHealthSnapshot_t snap;
    float temp_degC;
    float batt_pct;

    BoardHealthStatus_t temp_st;
    BoardHealthStatus_t batt_st;

    temp_st = BoardHealth_ReadTemperature(&temp_degC);
    batt_st = BoardHealth_ReadBattery(&batt_pct);

    uint32_t now = osKernelGetTickCount();
    snap.task_last_run_ms = now;

    if (temp_st == BOARD_HEALTH_OK)
    {
        snap.temperature_degC = temp_degC;
        snap.temp_last_valid_ms = now;
    }

    if (batt_st == BOARD_HEALTH_OK)
    {
        snap.battery_pct = batt_pct;
        snap.batt_last_valid_ms = now;
    }

    BoardHealthSnapshot_Write(&snap);
}

```

Figure 22: Step del task dedicato alla lettura della batteria e temperatura.

Per garantire l'affidabilità a lungo termine, il sistema integra un modulo di diagnostica basato su convertitori Analogico-Digitali (ADC) che monitorano costantemente i parametri vitali della scheda. Questo modulo si occupa di due aspetti critici:

- **Gestione della Batteria:** Il sistema legge la tensione della batteria attraverso un partitore resistivo. Poiché la scarica di una batteria non è lineare, il software utilizza una *Look-Up Table* (LUT) e

un'interpolazione lineare per convertire i Volt in una percentuale di carica residua (0-100%). Ciò fornisce all'utente un'indicazione realistica dell'autonomia residua.

- **Monitoraggio Termico:** Utilizzando il sensore di temperatura interno al microcontrollore STM32G4, il sistema calcola la temperatura operativa della logica di controllo. Il calcolo sfrutta i dati di calibrazione salvati in fabbrica dal produttore per garantire la massima precisione.

Per evitare letture errate dovute a disturbi elettrici o picchi temporanei, i dati non vengono usati così come sono, ma vengono processati attraverso una media mobile. Questo filtro "ammorbidisce" le letture su una finestra di 10 campioni, garantendo che i valori visualizzati siano stabili e puliti. Infine, il modulo verifica che i valori rientrino in range di sicurezza (es. 7V - 15V per la batteria), segnalando errori di sistema qualora i parametri diventino critici

4.3.2 Task: supervisore Board 1

Il supervisore della Board 1 è implementato tramite una macchina a stati sviluppata in Simulink Stateflow. Questo task ha il compito di supervisionare l'integrità dell'hardware e di assegnare i riferimenti di velocità lineare e di rotazione.

In Figura 23 è illustrato lo step del task del supervisore della Board 1, con il dettaglio delle interfacce di input e output.

```

void Supervisor_TaskStep(void)
{
    static BoardHealthSnapshot_t bh;
    static EncoderSnapshot_t enc;
    static RxSnapshot_t rx;
    static SupervisorSnapshot_t sup;
    static uint8_t last_sup_counter;
    static uint32_t last_sup_update_ms;

    BoardHealthSnapshot_Read(&bh);
    EncoderSnapshot_Read(&enc);
    RxSnapshot_Read(&rx);

    uint32_t now = osKernelGetTickCount();

    CommPayloadB2_t payload = rx.payload;

    if (payload.alive_counter != last_sup_counter) {
        last_sup_counter = payload.alive_counter;
        last_sup_update_ms = now;
    }

    SupervisorB1_U.Board2_Data = rx;
    SupervisorB1_U.Board_Health = bh;
    SupervisorB1_U.Encoder = enc;
    SupervisorB1_U.last_valid_b2_ms = last_sup_update_ms;
    SupervisorB1_U.now_ms = now;

    SupervisorB1_step();

    sup.critical_mask = SupervisorB1_Y.critical_mask;
    sup.degraded_mask = SupervisorB1_Y.degraded_mask;
    sup.speed_ref_rpm = SupervisorB1_Y.v_ref;
    sup.steering_cmd = SupervisorB1_Y.omega_ref;

    sup.current_action = SupervisorB1_Y.current_action;
    sup.isBoardActuating = !SupervisorB1_Y.give_b2_actuation;
}

```

Figure 23: Step del task del supervisore della Board 1.

Il task elabora in tempo reale i dati provenienti dalla **Board Health** (stato di batteria e temperatura), i feedback degli **Encoder** motori e i pacchetti dati ricevuti dalla Board 2.

La logica di controllo è strutturata su tre pilastri fondamentali:

- **Diagnostica e Fault Masking:** il sistema valuta l'integrità dei dati e la persistenza della comunicazione con la Board 2. In base alla latenza

dei messaggi e alla coerenza dei sensori, può far transitare il rover in stati di operatività *Degraded* o *Critical*. Questi stati permettono al supervisore di inibire manore o di limitare la potenza alle ruote in caso di anomalie, garantendo maggiore protezione.

- **Gestione delle Manovre Complesse:** A differenza della Board 2, focalizzata sull’evitamento ostacoli, la Board 1 gestisce comandi di alto livello come la *Rotazione a 180 gradi* assistita e la procedura di *Emergency Stop*, monitorando l’arresto effettivo delle ruote tramite i sensori di velocità.
- **Arbitraggio dell’Attuazione:** Il supervisore valuta costantemente l’affidabilità della comunicazione inter-board. Qualora rilevi un’instabilità nella comunicazione con l’altra scheda, è in grado di cedere i privilegi di attuazione alla Board 2, realizzando una strategia di ridondanza funzionale che aumenta la resilienza del rover. I motivi per cui la Board 1 può decidere di cedere il controllo alla Board 2 quando la comunicazione con quest’ultima si interrompe o si degrada derivano dal fatto che Board 2 ha a disposizione i sonar e soprattutto il comando utente, fondamentali per decider dove andare e se deviare il percorso o addirittura fermarsi.

Lo snapshot del supervisore di Board 1 è mostrato in Figura 24.

```
/* ===== Snapshot Supervisore Board 1 ===== */
typedef struct
{
    /* Timestamp decisione */
    uint32_t task_last_run_ms;

    /* Fault di board 1 presenti che comportano uno stato degradato*/
    uint32_t degraded_mask;

    /* Fault di board 1 presenti che comportano uno stato di emergenza*/
    uint32_t critical_mask;

    /* Comandi AUTORIZZATI */
    float speed_ref_rpm;      /* velocità longitudinale */
    float steering_cmd;       /* comando sterzata */

    SupervisorCommand_t current_action;
    bool isBoardActuating;

    /* Heartbeat / alive counter */
    uint8_t alive_counter;
} SupervisorSnapshot_t;
```

Figure 24: Step del task del supervisore della Board 1.

4.3.3 Task: controllo motori

Il task di controllo presente nella Board 1 è responsabile dell'attuazione dei comandi di movimento del rover, basandosi sui riferimenti di velocità lineare e angolare forniti dal supervisore. Il task quindi:

- legge i riferimenti di velocità lineare (v_{ref}) e angolare (ω_{ref}) calcolati dal supervisore.
- legge le velocità delle ruote attraverso gli encode.
- applica un controllo PID per regolare la potenza erogata ai motori, al fine di raggiungere i riferimenti di velocità desiderati.

Alla lettura degli encoder per stimare la velocità delle ruote è accompagnata una logica che permette il rilevamento di un'anomalia derivante dal fatto che si sta alimentando il motore ma gli encoder non rilvano movimenti, quindi la velocità calcolata è nulla. In particolare, se si rileva quest'anomalia per 20 cicli di esecuzione, si imposta la variabile $has_no_feedback(i)$ associata al motore di riferimento a 1.

Se uno o più encoder si rompono, il rover non deve smettere di funzionare, ma deve "stimare" la velocità mancante usando quella delle ruote sane. Il codice applica una sostituzione simmetrica:

- *1 Fallimento*: Se si rompe l'encoder anteriore sinistro, il sistema "copia" il valore di quello posteriore sinistro.
- *2 Fallimenti*: Se si rompe un intero lato, copia i valori dall'altro lato.
- *3 Fallimenti*: Tutte le ruote assumono il valore dell'unica ruota superstite.

Struttura del PI gerarchico del Rover Il controllo del rover si basa su un PI gerarchico creato a partire da quattro controllori PI distinti associati a ogni ruota del rover. Tra i controllori creati viene scelto, come base della legge di controllo, il PI associato al motore della ruota destra posteriore, questo perchè dalle analisi effettuate si evince essere la ruota con la dinamica più lenta tra le quattro. Una volta scelto il controllore di base, dunque, si procede ad implementare la legge di controllo gerarchica, che consente di controllare contemporaneamente tutti e quattro i motori del rover garantendo che questi raggiungano insieme la velocità desiderata, dettata dall'utente tramite il joystick.

La legge di controllo è organizzata secondo una struttura a quattro livelli principali, ciascuno basato su un regolatore PI dedicato:

1. controllo globale della velocità del rover;
2. sincronizzazione tra asse anteriore e asse posteriore;
3. controllo differenziale dell'asse posteriore;
4. controllo differenziale dell'asse anteriore.

Questa organizzazione consente di separare chiaramente:

- il controllo della velocità longitudinale;
- la sincronizzazione tra gli assi;
- la gestione della rotazione del rover.

Controllo globale della velocità Il livello più alto della gerarchia è rappresentato dal PI globale, che ha il compito di regolare la velocità media complessiva del rover. La velocità globale viene calcolata come media delle velocità delle quattro ruote.

Il regolatore PI globale calcola l'errore di velocità come differenza tra il riferimento richiesto dall'utente e la velocità globale misurata, e utilizza tale errore per generare un comando di base comune a tutti i motori:

$$u_{base} = PI(\omega_{ref} - \omega_{global})$$

Il PI globale è rappresentato dal PI che è stato scelto come base del controllo gerarchico, identificato dal controllore progettato in precedenza ed associato alla ruota destra posteriore.

Sincronizzazione tra asse anteriore e asse posteriore Una volta generata l'uscita da parte del PI globale essa non viene direttamente applicata a tutti i motori, questo perché i motori, essendo diversi, presentano una risposta al gradino diversa, hanno tempi di salita e assestamento differenti nonché velocità massime raggiungibili differenti. Se usassimo direttamente l'uscita del PI globale su tutti i motori essi non sarebbero sincronizzati. Per questo motivo si utilizza un secondo livello di controllo, dedicato alla sincronizzazione tra asse anteriore e asse posteriore.

Tale livello consiste nell'utilizzare un regolatore PI dedicato che agisce sull'errore definito come differenza tra la velocità media dell'asse anteriore e quella dell'asse posteriore:

$$e_{asse} = \omega_{anteriore} - \omega_{posteriore}$$

Il termine di correzione generato dal PI punta a portare a zero l'errore così rappresentato. Tale termine viene:

- sommato al comando dell'asse posteriore, in modo da aumentare la velocità delle ruote posteriori quando l'asse anteriore risulta più veloce (errore positivo) e ridurla nel caso opposto (errore negativo);
- sottratto al comando dell'asse anteriore, in modo da ridurre la velocità delle ruote anteriori quando esse risultano più veloci e aumentarla nel caso opposto.

In questo modo:

- entrambi gli assi tendono a mantenere la stessa velocità media;
- vengono compensate eventuali differenze di carico o attrito;
- si garantisce un moto rettilineo stabile del rover.

Controllo differenziale e gestione della rotazione Una volta ottenuta l'uscita per ruote posteriori e anteriori è necessario utilizzare un ulteriore livello di controllo. In particolare la gerarchia prevede due livelli inferiori di controllo, uno dedicato alle ruote posteriori ed uno per quelle anteriori, responsabili del controllo differenziale.

Il controllo differenziale ha il compito di:

- bilanciare la velocità tra ruota destra e sinistra su ogni asse;
- generare la rotazione del rover attorno al proprio baricentro in risposta ai comandi di sterzo.

Tramite questi livelli di controllo il rover può girare su sé stesso, effettuare sterzate e mantenere un traiettoria rettilinea durante la marcia, compensano eventuali differenza di velocità tra il lato sinistro e quello destro.

L'errore differenziale viene definito come:

$$e_{diff} = (\omega_{dx} - \omega_{sx}) - 2 \cdot \omega_{target}$$

dove ω_{target} è proporzionale al comando di sterzo fornito dall'utente.

I regolatori PI differenziali generano una correzione del tutto analoga a quella introdotta dal PI sugli assi, basata sull'errore di velocità tra i due lati, che viene:

- sottratta al comando della ruota destra;
- aggiunta al comando della ruota sinistra.

Tramite questo approccio è possibile aggiungere dei termini a destra o sinistra per realizzare una rotazione controllata del rover senza alterare la velocità media imposta dal livello globale.

Sistema di attuazione Il sistema di attuazione dei motori è realizzato mediante driver di potenza Sabertooth 2x12, impiegati per il pilotaggio dei motori DC del rover.

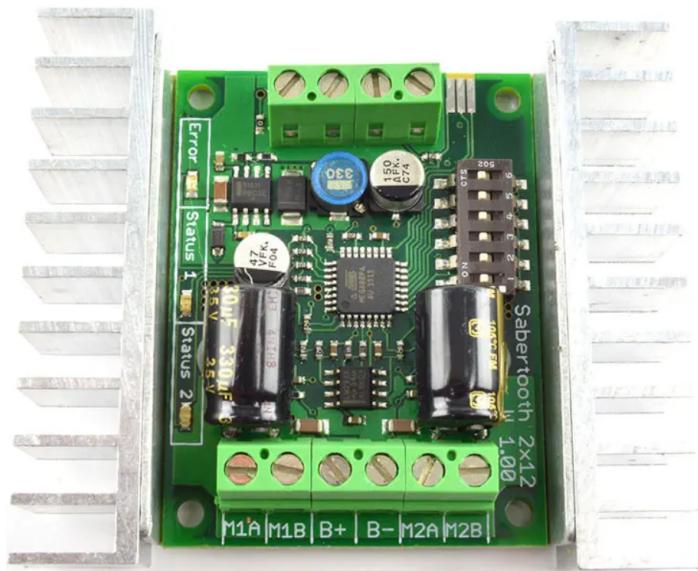


Figure 25: Driver di potenza Sabertooth 2x12

Nel sistema sono impiegati due driver Sabertooth, ciascuno responsabile del controllo di una coppia di motori:

- un driver dedicato alle ruote anteriori del rover;
- un driver dedicato alle ruote posteriori del rover.

Modalità di controllo dei driver I driver Sabertooth supportano diverse modalità di comando, tra cui il controllo analogico tramite PWM filtrato, la modalità *Simplified Serial*, la modalità con *Slave Select* e la modalità *Serial Packetized*. Tra queste, è stata scelta la modalità *Serial Packetized*. La scelta di tale modalità, al posto di un controllo PWM diretto, è motivata da diversi fattori progettuali:

- maggiore affidabilità del comando, grazie alla trasmissione digitale dei setpoint;
- gestione interna delle protezioni da parte del driver (sovrapotenza, sovrattensione, fault);
- riduzione della sensibilità ai disturbi elettrici;
- semplificazione del software di basso livello sul microcontrollore;
- riduzione del numero di linee di controllo necessarie tra microcontrollore e driver.

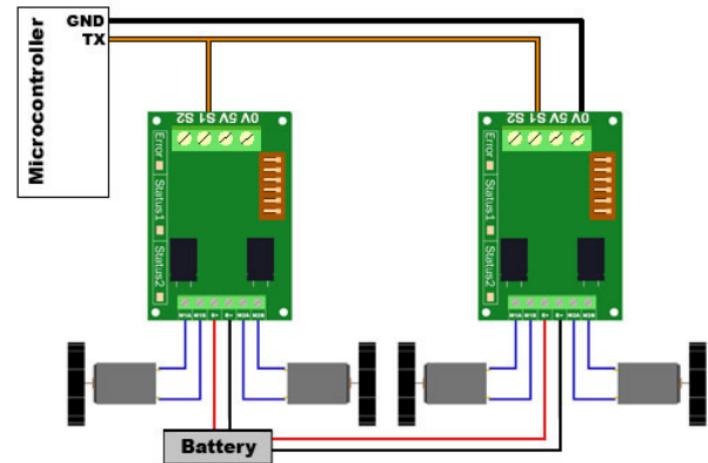


Figure 26: Collegamento MCU-Sabertooth in modalità Packetized serial

Tale modalità permette di inviare comandi strutturati come pacchetti da 4 byte contenenti indirizzo del driver, numero del motore, valore del setpoint di velocità e checksum, riducendo la probabilità di comandi errati dovuti a disturbi elettrici o rumore sulla linea.

Ogni Sabertooth è stata configurata con un indirizzo seriale distinto, impostato tramite i DIP switch hardware presenti sul driver:

- Sabertooth che controlla le ruote anteriori configurata con indirizzo 135;
- Sabertooth che controlla le ruote posteriori configurata con indirizzo 134.

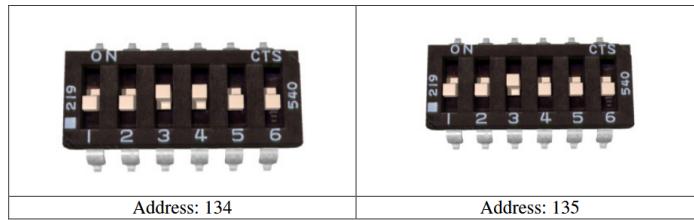


Figure 27: Impostazione dell’indirizzo del driver Sabertooth tramite DIP switch
4-5-6

L’assegnazione di indirizzi differenti permette di collegare entrambi i driver sulla stessa linea seriale. In questo modo, il microcontrollore può inviare, usando la stessa linea seriale, comandi selettivi ai motori anteriori o posteriori semplicemente variando l’indirizzo del pacchetto trasmesso.

Per quanto riguarda l’alimentazione, è stato abbassato lo switch 3 di ciascun driver, attivando la modalità *Lithium Cutoff*. In questa configurazione il driver Sabertooth rileva automaticamente il numero di celle della batteria al momento dell’accensione e imposta una soglia di spegnimento pari a 3.0V per cella, prevenendo la scarica eccessiva del pacco batterie e proteggendo l’hardware da condizioni di sotto tensione.

Struttura del pacchetto seriale Il protocollo Packetized Serial dei driver Sabertooth definisce un formato di pacchetto composto da quattro byte, utilizzato per la codifica dei comandi di controllo dei motori. La struttura del pacchetto è la seguente:

- Address byte: identifica il driver destinatario;
- Command byte: specifica il motore (1 o 2) e la direzione di rotazione;
- Data byte: rappresenta il setpoint di velocità nel range 0–127;
- Checksum: calcolato come somma dei tre byte precedenti mascherata a 7 bit.

<u>Packet</u>
Address: 135
Command : 0
Data: 64
Checksum: 72

Figure 28: Esempio di pacchetto Packetized serial letto dal driver Sabertooth con indirizzo 135, relativo al comando del motore M1 con setpoint di velocità del 50% in direzione positiva

4.3.4 Task: log dei dati per il debug

La Board 1 integra un task dedicato alla stampa dei dati per scopi di debug. Questo task, eseguito a cadenza regolare, acquisisce gli snapshot:

- **RxSnapshot**
- **EncoderSnapshot**
- **BoardHealthSnapshot**

e stampa i dati più rilevanti su console. Questa funzionalità è fondamentale per monitorare lo stato del sistema durante le fasi di sviluppo e test, permettendo di identificare rapidamente eventuali anomalie o comportamenti imprevisti. Lo step del task è mostrato in Figura 29, in cui vengono evidenziati i dati acquisiti e stampati su console.

```

void Log_TaskStep(void)
{
    static char log_buf[LOG_BUF_LEN];

    BleControllerSnapshot_t ble;
    IMUSnapshot_t imu;
    SonarSnapshot_t sonar;
    RxSnapshot_t rx;

    BleControllerSnapshot_Read(&ble);
    IMUSnapshot_Read(&imu);
    SonarSnapshot_Read(&sonar);
    RxSnapshot_Read(&rx);

    Log_FormatSnapshot(log_buf, LOG_BUF_LEN,
                       &ble, &imu, &sonar, &rx);

    printf("%s", log_buf);
}

```

Figure 29: Step del task dedicato alla stampa dei dati per scopi di debug.

4.3.5 Task: led

Per fornire un feedback immediato sullo stato operativo del sistema senza l’ausilio di terminali esterni, è stato implementato un task dedicato alla gestione della diagnostica visiva tramite LED. Il modulo `led_task.c` monitora le maschere di errore e le azioni correnti decise dai supervisori di entrambe le board, traducendole nel seguente protocollo di segnalazione:

Colore LED	Comportamento	Significato Operativo
Rosso	Fisso	Guasto Critico (Sistema bloccato)
Rosso	Lampeggiante	Guasto Degraded (Prestazioni ridotte)
Giallo (L/R)	Fisso	Manovra di schivata o sterzata in corso
Blu	Fisso	Emergency Stop attivo (Frenata di emergenza)

Table 1: Protocollo di segnalazione visiva del rover.

Questa implementazione risulta fondamentale durante i test sul campo, poiché permette di distinguere istantaneamente tra un normale comportamento di evitamento ostacoli (LED Giallo) e un arresto dovuto a un’anomalia hard-

ware o software (LED Rosso).

4.4 Task di trasmissione, ricezione e protocollo di comunicazione tra le due board

Le due board comunicano tra loro tramite un collegamento seriale UART. Su entrambe le board sono presenti due task distinti:

- un task di trasmissione, eseguito periodicamente con periodo di 20 ms;
- un task di ricezione di tipo event-driven, attivato dall'arrivo di nuovi dati sulla UART tramite interrupt.

Il task di trasmissione invia periodicamente un insieme di informazioni rappresentative dello stato locale della board, mentre il task di ricezione ricostruisce lo stato remoto a partire dai dati ricevuti.

4.4.1 Scelta di comunicazione asincrona

La comunicazione tra le due board è di tipo asincrono. In particolare, ciascuna board trasmette i propri dati in modo indipendente, in base alla propria temporizzazione locale, senza richiedere una sincronizzazione esplicita con l'altra board. Questo significa che non esiste un istante preciso in cui entrambe le board trasmettono simultaneamente, ma ciascuna invia i propri dati quando viene eseguito il rispettivo task periodico di trasmissione.

Questa scelta progettuale consente di evitare meccanismi di sincronizzazione bloccanti tra le due board. In particolare, una board non deve attendere che l'altra sia pronta a trasmettere o ricevere, né è necessario interrompere l'esecuzione dei task dell'altra board per forzare una comunicazione. In questo modo, la comunicazione non introduce ritardi o blocchi nell'esecuzione delle altre funzionalità del sistema.

L'approccio asincrono introduce un ritardo limitato tra la generazione delle informazioni su una board e la loro ricezione da parte dell'altra. Tuttavia, questo ritardo è trascurabile rispetto alle dinamiche del sistema, in particolare rispetto alla velocità di movimento del rover e alla frequenza con cui vengono prese le decisioni di controllo. La frequenza di trasmissione adottata è quindi sufficiente a garantire una visione aggiornata dello stato remoto e

un comportamento corretto del sistema.

Infine, evitare meccanismi di comunicazione bloccanti contribuisce a migliorare la sicurezza complessiva del sistema, evitando che la comunicazione ritardi l'eventuale esecuzione di funzionalità critiche sulle board. L'approccio adottato consente invece a ciascuna board di continuare a operare in modo autonomo e reattivo, garantendo che le funzioni di supervisione e controllo vengano eseguite nei tempi previsti.

4.4.2 Ricostruzione dello stato globale tramite lo scambio dati

Lo scopo della comunicazione tra le due board è permettere a ciascuna di ricostruire una visione coerente dello stato complessivo del sistema, utilizzando le informazioni ricevute dalla board remota. A tal fine, ogni board trasmette periodicamente un insieme selezionato di informazioni rappresentative del proprio stato locale.

Le informazioni trasmesse non corrispondono necessariamente ai dati grezzi acquisiti dai sensori, ma consistono principalmente in una rappresentazione sintetica e già elaborata dal supervisore. In particolare, vengono trasmesse informazioni di stato e indicatori di fault che permettono alla board ricevente di determinare se i sottosistemi remoti stanno operando correttamente, in modalità degradata o in condizioni critiche. Questo consente di ridurre la quantità di dati trasmessi e semplificare l'integrazione delle informazioni remote nel processo decisionale locale.

I dati grezzi vengono invece trasmessi solo quando sono necessari per il funzionamento del sistema o per la corretta interpretazione dei comandi. In questo modo, ciascuna board riceve tutte le informazioni essenziali per prendere decisioni corrette, senza richiedere l'accesso diretto a tutti i dettagli interni della board remota.

In Figura 30 è mostrata l'interazione tra il task di trasmissione della Board 2 e il task di ricezione della Board 1. In particolare, la Board 2 trasmette:

- Informazioni sullo stato del supervisore, che includono sia gli indicatori di fault sia il comando di sicurezza corrente. Questo permette alla board ricevente di conoscere se la board remota si trova in condizioni normali, degradate o critiche, e quale azione di sicurezza è stata eventualmente richiesta, come arresto di emergenza o manovre di evitamento ostacolo (Supervisor snapshot);

- Informazioni relative ai comandi utente, necessarie a determinare il comportamento richiesto del rover, come velocità, direzione e attivazione di comandi specifici (Ble snapshot);
- Informazioni sull'orientamento del rover, utilizzate per monitorare e verificare il completamento di manovre controllate, come l'inversione di direzione (Imu snapshot).

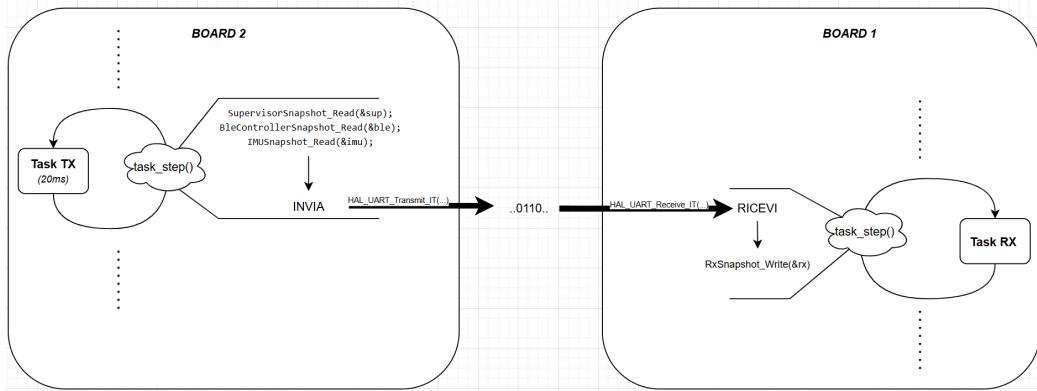


Figure 30: Interazione tra il task di trasmissione della Board 2 e il task di ricezione della Board 1.

Allo stesso modo, nella Figura 31 è mostrata l'interazione tra il task di trasmissione della Board 1 e il task di ricezione della Board 2. In questo caso, la Board 1 trasmette le informazioni locali relative ai sottosistemi di cui è responsabile, permettendo alla Board 2 di ricostruire una visione coerente dello stato globale del rover.

Vengono trasmesse le informazioni sullo stato del supervisore locale tramite indicatori di fault che permettono di valutare lo stato operativo della Board 1 e integrarlo nella determinazione dello stato globale (Supervisor snapshot). Inoltre, vengono trasmesse le velocità delle ruote misurate dagli encoder (Encoder snapshot), che permettono alla Board 2 di verificare il comportamento del rover e di controllare il corretto arresto dei motori durante condizioni di sicurezza come la frenata di emergenza.

Come nel caso della Board 2, vengono trasmesse solo le informazioni strettamente necessarie alla supervisione distribuita, utilizzando una rappresentazione sintetica dello stato per ridurre il carico di comunicazione e garantire un coordinamento efficace tra le due board.

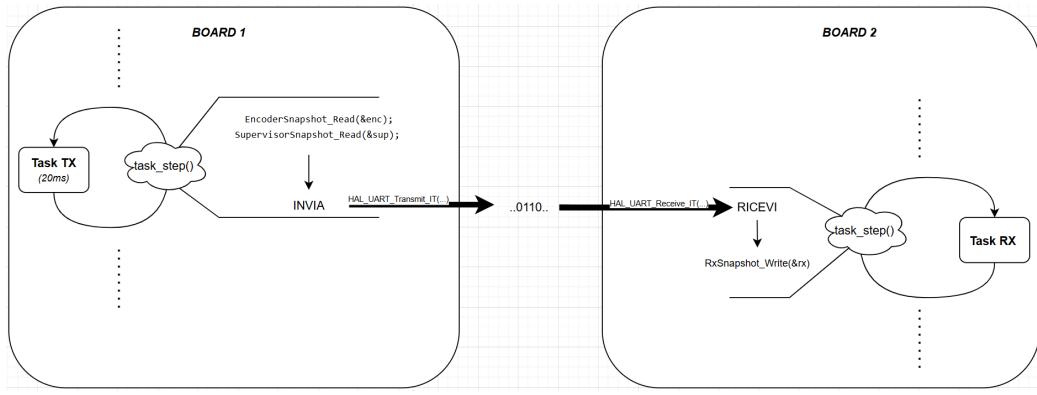


Figure 31: Interazione tra il task di trasmissione della Board 1 e il task di ricezione della Board 2.

4.4.3 Struttura del protocollo di comunicazione

La comunicazione tra le due board utilizza un protocollo basato su frame di lunghezza fissa, progettato per garantire sincronizzazione, integrità dei dati e robustezza rispetto a errori di trasmissione.

Ogni frame è composto da tre sezioni principali:

- un header, che contiene le informazioni di controllo del frame e un identificatore utilizzato per la sincronizzazione;
- un payload, che include le informazioni rappresentative dello stato della board trasmittente;
- un campo CRC (*Cyclic Redundancy Check*), utilizzato per verificare l'integrità dei dati ricevuti.

```

/* ===== Common frame header ===== */
typedef struct __attribute__((packed))
{
    uint16_t msg_id;
    uint16_t payload_len;
    uint16_t seq;
    uint32_t timestamp_ms;
} CommFrameHeader_t;

/* ===== Payload Board1 -> Board2 ===== */
typedef struct __attribute__((packed))
{
    float wheel_speed_rpm[4];
    uint32_t degraded_mask;
    uint32_t critical_mask;
    uint32_t alive_counter;
} CommPayloadB1_t;

typedef struct __attribute__((packed))
{
    CommFrameHeader_t header;
    CommPayloadB1_t payload;
    uint16_t crc;
} CommFrameB1_t;

```

Figure 32: Pacchetto inviato da board 1.

Sincronizzazione del flusso dati L’identificatore presente nell’header `msg_id` consente al ricevitore di riconoscere l’inizio di un nuovo frame valido all’interno del flusso continuo di byte ricevuti sulla UART. Il task di ricezione analizza il flusso dati e ricerca questo valore per allinearsi correttamente con la struttura del frame.

Questo meccanismo consente di ristabilire automaticamente la sincronizzazione anche in presenza di errori di trasmissione, perdita di byte o disallineamenti del flusso dati (comunicazione fuori fase). In tali situazioni, il ricevitore continua a scorrere i dati fino a individuare un nuovo identificatore valido, permettendo la ripresa corretta della comunicazione senza richiedere un reset del sistema.

Una volta individuato l’inizio del frame, il ricevitore acquisisce l’intero messaggio e ne verifica l’integrità tramite il CRC. Quando un frame valido viene ricevuto, il payload viene salvato come nuovo stato della board remota e viene

registrato il timestamp di ricezione, che permette al supervisore di rilevare eventuali problemi relativi alla comunicazione.

Gestione dei fault di comunicazione La gestione di eventuali anomalie di comunicazione è affidata al supervisore, che monitora la frequenza di ricezione dei frame validi confrontandola con quella attesa.

Se i frame continuano ad arrivare ma con una frequenza inferiore a quella nominale, la comunicazione viene considerata degradata, se invece non viene ricevuto alcun frame valido per un intervallo di tempo superiore a un timeout prestabilito, la comunicazione viene dichiarata in stato critico.

5 Algoritmo di schedulazione dei task

Per la gestione dei task su entrambe le board, è stato adottato l'algoritmo di scheduling **Rate Monotonic (RM)**. RM è un algoritmo di scheduling a priorità fissa di tipo preventivo (preemptive). La scelta è motivata dal fatto che RM è l'algoritmo **ottimo** tra quelli a priorità statica: se un set di task non è schedulabile con RM, non lo sarà con nessun altro algoritmo a priorità fissa. Secondo la logica RM, la priorità di un task è inversamente proporzionale al suo periodo:

- Task con periodi più brevi (alta frequenza) → Priorità più alta.
- Task con periodi più lunghi (bassa frequenza) → Priorità più bassa.

5.1 Calcolo del Fattore di Utilizzo

Il primo passo per validare il sistema consiste nel calcolare il fattore di utilizzo della CPU (U) per ogni board. Questo valore rappresenta la frazione di tempo in cui il processore è occupato nell'esecuzione dei task applicativi ed è definito come:

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \quad (1)$$

dove C_i è il WCET misurato e T_i è il periodo del task i -esimo.

5.2 Criterio di Schedulabilità: Limite di Liu-Layland

Per garantire che tutti i task rispettino le proprie scadenze (deadline), è stato applicato il **test di condizione sufficiente** di Liu-Layland. Un set di n task è sicuramente schedulabile, se il fattore di utilizzo totale U è inferiore o uguale a un limite superiore U_{lub} (Least Upper Bound), calcolato come:

$$U \leq n(2^{1/n} - 1) \quad (2)$$

Per il nostro sistema, dove sono presenti 7 task in esecuzione su entrambe le board, i limiti teorici sono:

$$U_{lub} \leq 7(2^{1/7} - 1) \approx 0.72(72.0\%). \quad (3)$$

Se l'utilizzo calcolato U risulta inferiore a questi valori, il sistema è sicuramente schedulabile utilizzando RM.

5.3 Verifica Numerica dei task su Board 1

Sulla base dei WCET acquisiti tramite il registro DWT, i parametri risultanti per i task di Board 1 sono:

Task (Board 1)	Periodo (T_i)	WCET (C_i)	Utilizzo (U_i)
Task_Control	10 ms	148.71 μ s	1.487 %
Task_Supervisor	20 ms	137.55 μ s	0.688 %
Task_TX	20 ms	128.82 μ s	0.644 %
Task_BattTemp	100 ms	32.88 μ s	0.033 %
Task_RX	20 ms	90.54 μ s	0.453 %
Task_Led	100 ms	41 μ s	0.041 %
Task_Log	1000 ms	4989.35 μ s	0.499 %
Totale Board 1		U_{B1}	3.845 %

Table 2: Analisi di schedulabilità per la Board 1.

5.4 Verifica Numerica dei task su Board 2

Sulla base dei WCET acquisiti tramite il registro DWT, i parametri risultanti per i task di Board 2 sono:

Task (Board 2)	Periodo (T_i)	WCET (C_i)	Utilizzo (U_i)
Task_ReadBLE	20 ms	5022.88 μ s	25.11 %
Task_Supervisor	20 ms	332.66 μ s	1.66 %
Task_ReadSonars	60 ms	56.47 μ s	0.09 %
Task_ReadIMU	20 ms	5013.58 μ s	25.07 %
Task_TX	20 ms	120.01 μ s	0.60 %
Task_RX	20 ms	69.14 μ s	0.35 %
Task_Log	1000 ms	5129.35 μ s	0.51 %
Totale Board 2		U_{B2}	53.39 %

Table 3: Analisi di schedulabilità per la Board 2.

5.5 Assegnazione delle priorità

Una volta appurato che il set di task fosse schedulabile secondo l'algoritmo Rate Monotonic, in base ai periodi di ogni task sono state assegnate le seguenti priorità:

The screenshot shows a configuration interface for FreeRTOS tasks. At the top, there are four checkboxes: 'Tasks and Queues' (checked), 'Timers and Semaphores' (unchecked), 'Config parameters' (checked), and 'Include parameters' (checked). Below this, a table titled 'Tasks' lists seven tasks with their assigned priorities:

Task Name	Priority
Task_BattTemp	osPriorityBelowNormal
Task_TX	osPriorityNormal
Task_RX	osPriorityNormal
Task_Supervisor	osPriorityNormal
Task_Control	osPriorityHigh
Task_Log	osPriorityLow
Task_Led	osPriorityBelowNormal

Figure 33: Priorità assegnate ai task di Board 1 tramite FREERTOS

The screenshot shows a configuration interface for FreeRTOS tasks. At the top, there are four checkboxes: 'Tasks and Queues' (checked), 'Timers and Semaphores' (unchecked), 'Config parameters' (checked), and 'Include parameters' (checked). Below this, a table titled 'Tasks' lists seven tasks with their assigned priorities:

Task Name	Priority
Task_ReadBLE	osPriorityHigh
Task_ReadIMU	osPriorityHigh
Task_ReadSonars	osPriorityNormal
Task_Rx	osPriorityHigh
Task_Tx	osPriorityHigh
Task_Supervisor	osPriorityHigh
Task_Log	osPriorityLow

Figure 34: Priorità assegnate ai task di Board 2 tramite FREERTOS

6 Gestione della concorrenza e sincronizzazione: i Mutex

In un sistema real-time basato su task concorrenti che condividono risorse hardware o strutture dati, la sola gestione delle priorità tramite politiche di scheduling come il Rate Monotonic non è sufficiente a garantire il corretto funzionamento del sistema. Quando più task accedono in modo concorrente alle stesse risorse, è infatti necessario introdurre meccanismi di sincronizzazione per prevenire il fenomeno delle *race condition* e garantire la consistenza dei dati.

Nel progetto del rover, la sincronizzazione tra i task è stata realizzata principalmente mediante l'utilizzo di mutex forniti da FreeRTOS, usati per proteggere l'accesso alle risorse condivise.

Perché utilizzare i Mutex L'uso dei mutex (Mutual Exclusion) è stato preferito all'uso dei semplici semafori binari per la gestione delle sezioni critiche per tre motivi tecnici fondamentali:

1. Ownership: Un mutex può essere rilasciato esclusivamente dal task che lo ha acquisito. Questo garantisce che l'accesso a una risorsa condivisa sia strettamente controllato e non possa essere rilasciato accidentalmente da task non coinvolti nella sezione critica.
2. Priority inheritance: I mutex di FreeRTOS implementano il meccanismo di priority inheritance, che consente di evitare il fenomeno della priority inversion. In particolare, se un task ad alta priorità tenta di accedere a una risorsa occupata da un task a priorità inferiore, quest'ultimo eredita temporaneamente una priorità più alta, così da completare rapidamente la sezione critica e rilasciare il mutex.
3. Robustezza: I mutex sono progettati per essere robusti: nel caso in cui un task termini mentre detiene un mutex, la risorsa viene comunque rilasciata, evitando situazioni di blocco permanente del sistema.

6.1 Snapshot come struttura condivisa

Nel progetto del rover, la comunicazione tra task concorrenti non avviene tramite lo scambio diretto di messaggi, ma mediante l'utilizzo di strutture dati condivise denominate *snapshot*.

Uno snapshot rappresenta una fotografia consistente dello stato di un determinato sottosistema e racchiude tutte le informazioni che un task desidera rendere disponibili al resto del sistema. A seconda del task che lo produce, uno snapshot può contenere:

- dati acquisiti da sensori (ad esempio IMU, encoder, sonar);
- informazioni di validità temporale dei dati, come l’istante dell’ultima acquisizione valida;
- dati ricevuti da periferiche esterne o da altre board (ad esempio tramite UART);
- variabili di stato o grandezze calcolate che devono essere esposte ad altri task, come parametri di controllo o informazioni per il supervisore.

In questo modo, ogni task produttore aggiorna periodicamente il proprio snapshot, mentre i task che necessitano di tali informazioni leggono sempre l’ultima versione completa dei dati tramite un’API dedicata, senza interagire direttamente con il meccanismo di sincronizzazione.

```

typedef struct
{
    float temperature_degC;          /* ultima lettura valida della temperatura */
    float battery_pct;              /* ultima lettura valida della batteria */

    uint32_t task_last_run_ms;      /* ultima esecuzione del task */

    uint32_t temp_last_valid_ms;    /* ultima rilevazione valida della temperatura */
    uint32_t batt_last_valid_ms;    /* ultima rilevazione valida della batteria */
} BoardHealthSnapshot_t;

/* Inizializzazione del mutex relativo allo snapshot Board Health */
void BoardHealthSnapshot_MutexInit(osMutexId_t mutex_handle);

/* Writer API (chiamata esclusivamente dal task Board Health) */
void BoardHealthSnapshot_Write(const BoardHealthSnapshot_t *src);

/* Reader API (chiamata dai task interessati a leggere i dati prodotti) */
void BoardHealthSnapshot_Read(BoardHealthSnapshot_t *dst);

```

Figure 35: Struttura dello snapshot Board_Health

6.1.1 Protezione degli snapshot tramite mutex

Poiché gli snapshot sono strutture dati condivise tra più task, il loro accesso deve essere opportunamente sincronizzato per garantire la coerenza delle informazioni lette. Per questo motivo, ad ogni snapshot è associato un mutex

dedicato, utilizzato per proteggere le operazioni di lettura e scrittura.

I mutex associati agli snapshot sono stati definiti staticamente tramite l’interfaccia CubeMX.

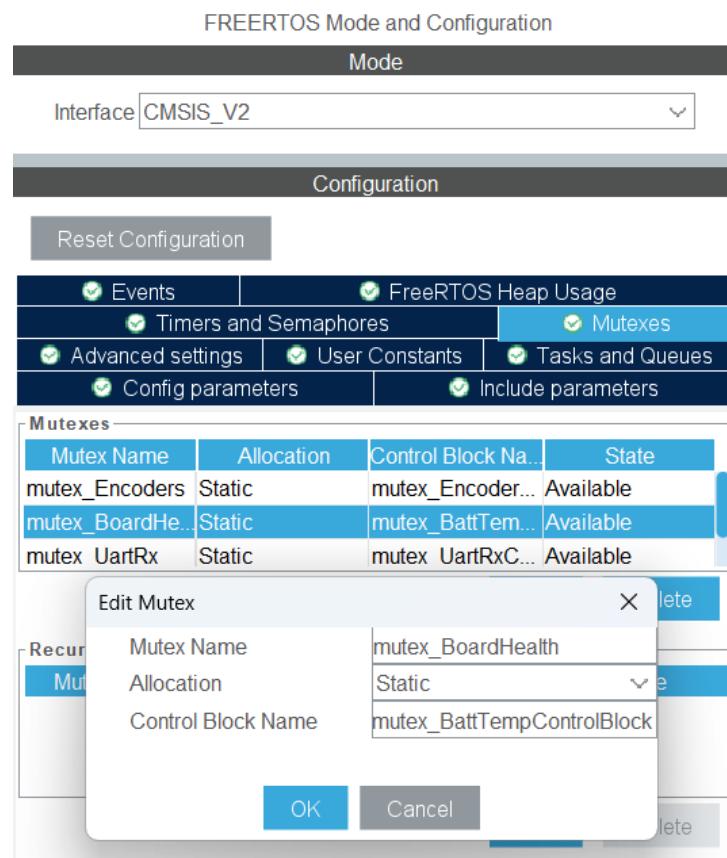


Figure 36: Configurazione CubeMX del mutex relativo allo snapshot Board_Health

Durante la fase di inizializzazione di FreeRTOS, gli handle dei mutex vengono passati ai moduli dei relativi snapshot, da cui viene poi utilizzato per sincronizzare l’accesso ai dati contenuti nello snapshot.

```

/* Definitions for mutex_BoardHealth */
osMutexId_t mutex_BoardHealthHandle;
osStaticMutexDef_t mutex_BattTempControlBlock;
const osMutexAttr_t mutex_BoardHealth_attributes = {
    .name = "mutex_BoardHealth",
    .cb_mem = &mutex_BattTempControlBlock,
    .cb_size = sizeof(mutex_BattTempControlBlock),
};

```

Figure 37: Definizione statica del mutex e dei relativi attributi generata da CubeMX

```

void MX_FREERTOS_Init(void) {
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Create the mutex(es) */

    /* creation of mutex_BoardHealth */
    mutex_BoardHealthHandle = osMutexNew(&mutex_BoardHealth_attributes);

    /* USER CODE BEGIN RTOS_MUTEX */
    BoardHealthSnapshot_MutexInit(mutex_BoardHealthHandle),
}

```

Figure 38: Inizializzazione del mutex in `MX_FREERTOS_Init` e passaggio dell'handle al modulo dello snapshot `Board_Health`

La scrittura di uno snapshot avviene in modo atomico: il task produttore acquisisce il mutex, aggiorna l'intera struttura e rilascia il mutex al termine dell'operazione. Analogamente, un task consumatore acquisisce il mutex prima di copiare localmente lo snapshot e lo rilascia subito dopo.

```

/* Istanza dello snapshot */
static BoardHealthSnapshot_t snapshot;

/* Mutex */
static osMutexId_t snapshot_mutex;

void BoardHealthSnapshot_MutexInit(osMutexId_t mutex_handle){
    if(snapshot_mutex == NULL){
        snapshot_mutex = mutex_handle;
    }
}

void BoardHealthSnapshot_Write(const BoardHealthSnapshot_t *src){
    if (src == NULL)
        return;

    osMutexAcquire(snapshot_mutex,osWaitForever);
    snapshot = *src; /* copia atomica della struttura */
    osMutexRelease(snapshot_mutex);
}

void BoardHealthSnapshot_Read(BoardHealthSnapshot_t *dst){
    if (dst == NULL)
        return;

    osMutexAcquire(snapshot_mutex,osWaitForever);
    *dst = snapshot; /* copia atomica della struttura */
    osMutexRelease(snapshot_mutex);
}

```

Figure 39: Implementazione dello snapshot Board Health

Questo schema, basato sull'uso congiunto di snapshot e mutex dedicati, permette di condividere i dati tra i task in modo semplice e controllato. L'adozione dello stesso approccio per tutti i sottosistemi del rover rende il codice più ordinato e facilita l'aggiunta di nuovi task o nuove funzionalità senza modificare l'architettura preesistente.

6.1.2 Mutex e snapshot implementati su Board 1

Gli snapshot gestiti su Board 1 contengono dati rilevanti per il funzionamento e la sicurezza del rover, come informazioni sullo stato della batteria e della temperatura, riferimenti utilizzati dal ciclo di controllo e segnalazioni di sicurezza generate dal supervisore, tra cui eventuali condizioni di arresto di emergenza. Per questo motivo dobbiamo assicurarci che i dati disponibili siano sempre coerenti.

Di seguito sono riportati i mutex utilizzati, descrivendo per ciascuno i task coinvolti e il ruolo dei dati protetti:

- **mutex_Encoders**: Protegge lo snapshot contenente i dati degli encoder. I dati vengono scritti dal task di controllo durante la fase di acquisizione

delle velocità delle ruote, mentre vengono letti dal task supervisore per verificare lo stato delle ruote e l'eventuale presenza di anomalie, inoltre, vengono letti anche dal task di trasmissione per inviare gli RPM alla Board 2, dove vengono utilizzati per diagnosi incrociate con i dati dell'IMU.

- **mutex_BoardHealth:** Protegge i dati relativi allo stato della board, come tensione della batteria e temperatura. Lo snapshot è scritto dal task di Board Health ed è letto dal task supervisore per il controllo dei limiti operativi e la gestione di eventuali condizioni critiche.
- **mutex_UartRx:** Protegge i dati ricevuti via UART dalla Board 2. Lo snapshot è scritto dal task di ricezione UART e letto dal task supervisore, che utilizza tali informazioni per valutare lo stato globale del rover e per determinare eventuali limitazioni sulla velocità o strategie di schivata ostacoli.
- **mutex_Supervisor:** Protegge lo snapshot del supervisore. I dati sono scritti dal task supervisore e letti dal task di controllo, che utilizza i riferimenti di velocità forniti e reagisce immediatamente a eventuali segnali di fault, arrestando o modificando il comportamento del sistema di attuazione.

6.2 Mutex e snapshot implementati su Board 2

Gli snapshot gestiti su Board 2 contengono i comandi provenienti dall'utente e dati acquisiti dall'ambiente esterno, come misure inerziali ed informazioni provenienti dai sensori a ultrasuoni, utilizzate per la gestione delle manovre di schivata e della frenata di emergenza. Per questo motivo dobbiamo assicurarci che i dati letti siano sempre coerenti tra task concorrenti.

Di seguito sono riportati i mutex utilizzati, descrivendo per ciascuno i task coinvolti e il ruolo dei dati protetti:

- **mutex_Ble:** Protegge lo snapshot contenente i comandi utente provenienti dal controller remoto. I dati vengono scritti dal task che legge l'interfaccia I^2C collegata alla scheda ESP32; mentre è letto dal task supervisore per verificare la freschezza dei dati e il corretto funzionamento della comunicazione, e dal task di trasmissione per inoltrare i setpoint alla Board 1, che aggiorna di conseguenza i riferimenti di controllo.

- **mutex_Imu**: Protegge lo snapshot dei dati inerziali acquisiti dall'IMU. I dati vengono scritti periodicamente dal task di lettura dell'IMU e sono letti dal task supervisore per verificare la correttezza dell'acquisizione, per effettuare controlli di coerenza con gli RPM ricevuti e per gestire le manovre di schivata o rotazione del rover. Lo snapshot è inoltre letto dal task di trasmissione per inviare lo yaw alla Board 1, che gestisce l'attuazione principale del sistema.
- **mutex_UartRx**: Protegge i dati ricevuti dalla Board 1 tramite UART. Lo snapshot è scritto dal task di ricezione UART e letto dal task supervisore, che utilizza tali informazioni per valutare lo stato globale del rover e determinare eventuali azioni correttive o di sicurezza da attuare anche su Board 2.
- **mutex_Sonar**: Protegge lo snapshot contenente le misure dei sensori a ultrasuoni. I dati sono scritti dal task di acquisizione dei sonar e letti dal task supervisore per la gestione delle manovre di schivata degli ostacoli o per l'attivazione della frenata di emergenza.
- **mutex_Supervisor**: Protegge lo snapshot del supervisore di Board 2. I dati sono scritti dal task supervisore e letti dal task di trasmissione, che invia alla Board 1 lo stato delle periferiche e le informazioni diagnostiche relative alla Board 2, nonché indicazioni di controllo per preservare la sicurezza del rover.

7 Condizioni di funzionamento

Il rover è dotato di due schede di controllo, denominate Board 1 e Board 2, che collaborano per garantire il funzionamento del sistema. In Figura 40 è mostrata l'architettura generale del sistema.

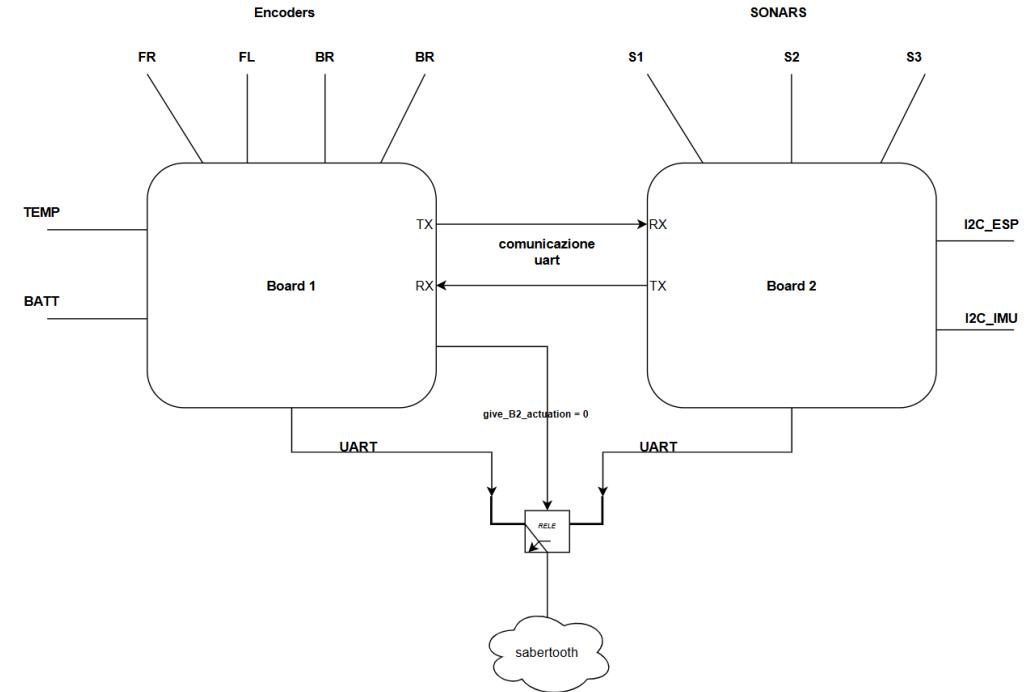


Figure 40: Architettura generale del sistema.

7.1 Condizioni di funzionamento nominale

In condizioni di funzionamento nominale la Board 1 è considerata il *Master*, in quanto si occupa dell'attuazione dei motori e della generazione dei riferimenti finali, inviando tramite UART i comandi di controllo agli attuatori. La Board 2 è invece considerata *Slave*, poiché non attua direttamente i motori ma fornisce informazioni di supporto al sistema di controllo.

Il riferimento di velocità lineare o angolare è calcolato a partire dai comandi utente inviati tramite controller BLE ad una scheda *ESP32* e successivamente ricevuti dalla Board 2 tramite protocollo *I2C*. La Board 2 trasmette quindi tali informazioni alla Board 1, che provvede ad applicare le opportune limitazioni di sicurezza e a generare il comando finale per l'attuazione.

Rilevamento ostacoli I riferimenti di velocità sono inoltre influenzati dalle condizioni ambientali rilevate dai sensori. In particolare, la Board 2, direttamente interfacciata con i sensori sonar, elabora le misure di distanza e genera opportuni comandi di sicurezza, come ad esempio `CMD_ESTOP`, `CMD_AVOID_LEFT` o `CMD_GO_LEFT`. Tali comandi vengono trasmessi alla Board 1 tramite comunicazione UART, dove il supervisore valuta lo stato corrente del sistema e, se necessario, modifica i riferimenti di moto garantendo il rispetto delle condizioni di sicurezza.

7.2 Modalità operative non nominali

Oltre alla modalità nominale, il sistema prevede due modalità degradate e una modalità critica, attivate dal supervisore in funzione dello stato delle periferiche, della qualità della comunicazione e della disponibilità delle due Board. Queste modalità garantiscono la sicurezza del rover anche in presenza di guasti o condizioni operative non ideali.

Modalità degradata con entrambe le Board operative Questa modalità si verifica quando una o più periferiche risultano operative ma con prestazioni ridotte, oppure quando parametri critici, come la qualità della comunicazione, la temperatura o il livello della batteria, si avvicinano a soglie di sicurezza.

In questa configurazione entrambe le Board restano attive e il sistema mantiene tutte le funzionalità di controllo e supervisione. Tuttavia, al fine di ridurre il rischio di instabilità o perdita di controllo, la velocità massima del rover viene limitata al 50% del valore nominale. Il sistema continua a utilizzare tutte le informazioni disponibili dai sensori e mantiene attive le normali strategie di sicurezza e supervisione del movimento.

Modalità degradata con Board isolata Questa modalità si verifica quando la Board 1 non riceve più aggiornamenti dalla Board 2 per un intervallo di tempo superiore alla soglia di sicurezza prevista. In questa condizione, la Board 1 cede i privilegi di attuazione alla Board 2, che assume il controllo diretto del rover.

Poiché in questa configurazione non è disponibile il normale livello di supervisione e coordinamento tra le due Board, il sistema opera con funzionalità di sicurezza conservative. In particolare, la velocità massima viene ridotta e, alla rilevazione di un ostacolo, viene eseguito un arresto immediato del rover,

senza effettuare manovre evasive. Questa modalità garantisce un comportamento fail-safe anche in assenza della piena cooperazione tra le due unità di controllo.

Modalità critica La modalità critica viene attivata quando il supervisore rileva un guasto che compromette la disponibilità di una funzionalità essenziale per il funzionamento sicuro del rover.

In questa condizione il sistema esegue un arresto di emergenza e disabilita l'attuazione dei motori, impedendo qualsiasi movimento. Il rover rimane in questo stato fino a quando la condizione di guasto non viene risolta e il sistema torna in uno stato operativo sicuro.

7.2.1 Esempio di funzionamento in modalità non nominale: fault e fallback encoder

Il rover è progettato per continuare a funzionare anche in presenza di guasti agli encoder delle ruote.

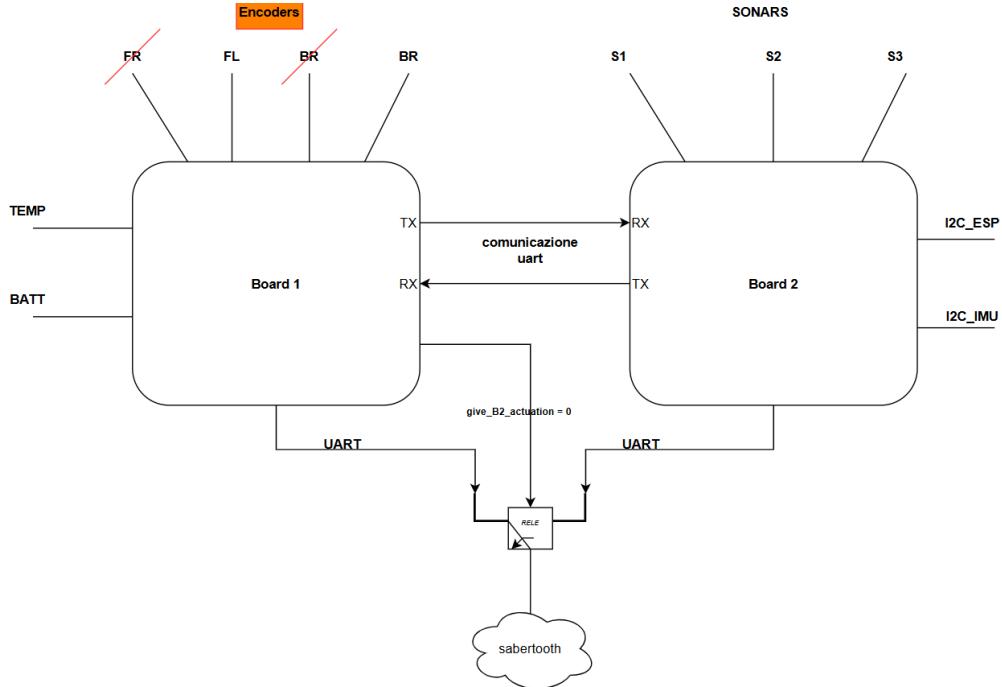


Figure 41: Logica di sostituzione in presenza di encoder non funzionanti.

Se uno o più encoder risultano non funzionanti, il sistema, quando è disponibile almeno un encoder valido, stima la velocità mancante utilizzando quella

delle ruote sane. Questa strategia consente di mantenere, quando possibile, il controllo in retroazione della velocità e quindi garantire accelerazioni e decelerazioni controllate. La logica di sostituzione è la seguente:

- *Un solo encoder guasto*: la velocità della ruota mancante viene sostituita con quella della ruota sullo stesso lato (anteriore o posteriore). In questa configurazione è ancora possibile sterzare.
- *Due encoder guasti*: il comportamento dipende dalla loro disposizione.
 - Se i due guasti interessano lo stesso lato del rover, le velocità del lato sano vengono replicate su quello guasto (per asse). Data la struttura del PI gerarchico risulta impossibile applicare la sterzata, in quanto, essendo le velocità di un lato copiate dall'altro, non è possibile dare velocità opposte ai due lati del rover. In questa configurazione è quindi disabilitata esplicitamente la sterzata.
 - Se invece i guasti sono distribuiti in modo da lasciare almeno un encoder funzionante per ciascun lato, la sterzata è ancora possibile.
- *Tre encoder guasti*: tutte le ruote assumono la velocità misurata dall'unico encoder superstite. In questo caso il rover può muoversi solo in modo approssimativamente rettilineo e la sterzata non è più realizzabile.
- *Quattro encoder guasti*: non essendo disponibile alcun feedback di velocità, il controllo in retroazione non è più applicabile e il sistema può operare esclusivamente in modalità open-loop.

La rilevazione di un encoder non funzionante avviene attraverso una logica che monitora la potenza erogata al motore e la velocità rilevata dagli encoder. Se si alimenta un motore ma gli encoder non rilevano movimenti (velocità calcolata è nulla) per un determinato numero di cicli di esecuzione, allora si imposta la variabile booleana *has_no_feedback(i)* associata al motore *i* a true, indicando che l'encoder è considerato non funzionante e attivando la logica di sostituzione descritta precedentemente.

Il supervisore della Board 1, rilevando questa condizione, porta la ruota associata nello stato *ENCODER_DEGRADED* e, di conseguenza, il rover entra in modalità degradata con entrambe le Board operative.

Se la Board 1 riceve dalla Board 2 una segnalazione di incoerenza del moto, indicata dal flag booleano *isMotionConsistent = false*, mentre uno o più

encoder risultano degradati, il supervisore considera la stima della velocità non più affidabile, assumendo un possibile guasto dell'attuatore associato. In questa condizione il sistema entra in modalità critica ed esegue un arresto di emergenza, disabilitando l'attuazione dei motori.

8 Supervisore Board 1

8.1 Panoramica generale

Il supervisore della Board 1 è implementato come un modulo Simulink denominato **SupervisorB1**, il cui schema a blocchi è illustrato in Figura 42.

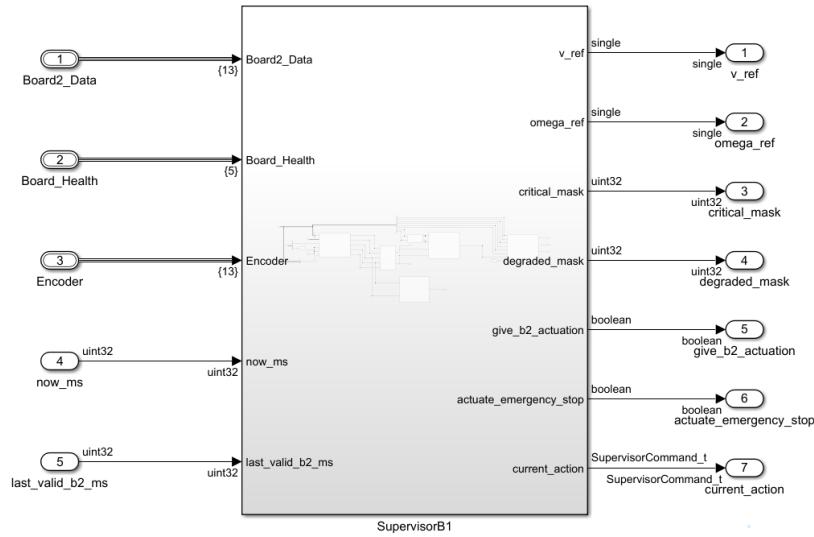


Figure 42: Schema a blocchi del modulo SupervisorB1.

Il suo compito è quello di determinare i riferimenti di velocità lineare (v_{ref}) e angolare (ω_{ref}) del rover, oppure di demandare l'attuazione alla Board 2, in funzione dell'elaborazione dei dati di input e delle condizioni di fault rilevate.

In particolare, esso è composto da quattro parti principali:

- **Rilevazione Faults locali:** si occupa di rilevare anomalie relative alla ricezione dei dati da Board2, agli encoder delle ruote ed ai sensori di temperatura e batteria, classificando lo stato di ciascuna periferica come OK, DEGRADED o CRITICAL.
- **Costruzione delle fault mask:** a partire dagli stati delle singole periferiche, costruisce le fault mask locali che rappresentano lo stato complessivo della Board 1. Queste vengono usate dalla Board 1, in combinazione con le fault mask ricevute dalla Board 2, per ricostruire lo stato globale del rover; inoltre vengono inviate alla Board 2, così che essa possa effettuare la stessa operazione.

- **Selezione della board attuante:** decide se la Board 1 deve continuare ad attuare o se il controllo deve essere passato alla Board 2, in funzione dello stato della comunicazione e delle condizioni di fault rilevate, garantendo che una sola board alla volta sia responsabile dell'attuazione.
- **Ricostruzione dello stato globale e calcolo dei riferimenti:** quando la Board 1 è responsabile dell'attuazione, combina le informazioni locali con quelle ricevute dalla Board 2 per determinare lo stato operativo del rover (ad esempio funzionamento normale, modalità degradata o arresto di emergenza) e calcolare i riferimenti di velocità lineare e angolare.

Nel seguito verranno descritti i segnali di input e output del supervisore, successivamente verranno descritte le quattro parti principali del supervisore descritte sopra.

8.2 Ingressi del supervisore

I segnali in input al supervisore sono i seguenti:

- **Board2_Data:** è l'ultimo snapshot di ricezione (*RxSnapshot*), aggiornato dal task di ricezione della Board 1. I segnali utilizzati dal supervisore sono:
 - *command*: rappresenta il comando generato dal supervisore della Board 2, che può assumere i seguenti valori:

```
typedef enum
{
    CMD_NORMAL = 0,
    CMD_ROTATE_180,
    CMD_GO_LEFT,
    CMD_GO_RIGHT,
    CMD_AVOID_RIGHT,
    CMD_AVOID_LEFT,
    CMD_STOP,
    CMD_ESTOP
} SupervisorCommand_t;
```

Figure 43: Comandi in uscita dal supervisore della Board 2.

- *x_norm* e *y_norm*: rappresentano i comandi normalizzati provenienti dal controller BLE e trasmessi dalla Board 2. I riferimenti

di velocità lineare e angolare vengono calcolati a partire da questi valori, in funzione dello stato del rover e del comando ricevuto.

- *btn1* e *btn2*: segnalano la pressione dei pulsanti presenti sul joystick e vengono utilizzati per attivare specifiche manovre, come la commutazione tra retromarcia e la rotazione di 180 gradi.
- *yaw*: rappresenta l’angolo di orientamento del rover, utilizzato dal supervisore per effettuare la rotazione di 180 gradi.
- *isMotionConsistent*: indica se il moto del rover è coerente con i comandi applicati, sulla base di una verifica effettuata dalla Board 2 utilizzando la IMU e gli encoder. Questo segnale viene utilizzato dalla Board 1 per effettuare un controllo incrociato: se un encoder non fornisce feedback e contemporaneamente il moto risulta non coerente, il supervisore può attribuire il fault al motore piuttosto che al sensore.
- *critical_mask* e *degraded_mask*: sono due maschere di errore calcolate dalla Board 2, in cui ogni bit rappresenta la presenza di una specifica anomalia critica o degradata. Queste informazioni vengono utilizzate per ricostruire lo stato globale del rover.
- *data_last_valid_ms*: rappresenta il tempo, in millisecondi, dell’ultimo dato valido ricevuto dalla Board 2, ed è utilizzato per rilevare eventuali timeout di comunicazione.

- **Board_Health**: è l’ultimo snapshot dello stato di salute della Board 1 (*BoardHealthSnapshot*), contenente informazioni quali temperatura e livello della batteria. La struttura dati è illustrata in Figura 61:

```
#typedef struct
{
    float temperature_degC;      /* ultima lettura valida della temperatura */
    float battery_pct;          /* ultima lettura valida della batteria */

    uint32_t task_last_run_ms;   /* ultima esecuzione del task */

    uint32_t temp_last_valid_ms; /* ultima rilevazione valida della temperatura */
    uint32_t batt_last_valid_ms; /* ultima rilevazione valida della batteria */
} BoardHealthSnapshot_t;
```

Figure 44: Struttura dati Board_Health.

- **Encoder**: è l’ultimo snapshot degli encoder delle ruote, contenente le velocità angolari delle ruote (*wheel_speed_rpm*) e un indicatore di eventuale assenza di feedback da uno dei motori (*has_no_feedback*). La struttura dati è illustrata in Figura 62:

```

@typedef struct
{
    float wheel_speed_rpm[4];
    bool hasNoFeedback[4];
        // indica se c'è corrispondenza tra comando e lettura encoder.
        // (è Vero se la lettura encoder restituisce 0 rpm in presenza
        // di un comando di velocità valido)

    uint32_t task_last_run_ms;      /* ultima esecuzione del task */
    uint32_t data_last_valid_ms[4]; /* ultimo istante in cui i dati acquisiti sono validi
                                    (uno per ogni encoder) */
} EncoderSnapshot_t;

```

Figure 45: Struttura dati Encoder.

- **now_ms**: rappresenta il tempo corrente in millisecondi, utilizzato per la gestione delle temporizzazioni e il rilevamento di timeout.
- **last_valid_b2_ms**: rappresenta il tempo, in millisecondi, dell'ultimo istante in cui la comunicazione con la Board 2 è stata considerata valida. Questo valore viene aggiornato quando vengono ricevuti nuovi dati coerenti dalla Board 2 ed è utilizzato dal supervisore per rilevare eventuali fault o relativi alla comunicazione.

8.3 Uscite del supervisore

I segnali in output che fornisce sono:

- **v_ref**: rappresenta il riferimento normalizzato di velocità lineare del rover, compreso nell'intervallo [-1, 1].
- **omega_ref**: rappresenta il riferimento normalizzato di velocità angolare del rover, compreso nell'intervallo [-1, 1].
- **critical_mask** e **degraded_mask**: sono due mask di errore che rappresentano le anomalie critiche e degradate rilevate dalla Board 1. Queste informazioni vengono utilizzate sia localmente sia trasmesse alla Board 2 per la ricostruzione dello stato globale del rover.
- **give_b2_actuation**: rappresenta un segnale booleano che indica se il controllo dell'attuazione deve essere passato alla Board 2.
- **actuate_emergency_stop**: rappresenta un segnale booleano che indica se deve essere attivato l'arresto di emergenza del rover.
- **current_action**: rappresenta l'azione attualmente eseguita dal supervisore.

La corrispondenza tra le fault mask e le anomalie rilevate è riportata in Tabella 4.

Bit	Mask	Descrizione dell'Anomalia
0	CRI/DEG	Temperatura fuori range o timeout dall'ultima lettura valida
1	CRI/DEG	Livello batteria fuori range o timeout dall'ultima lettura valida
2	CRI/DEG	Comunicazione con Board 2 lenta o in timeout
3	CRI/DEG	Fault della ruota front left: degradato se encoder non fornisce feedback, critico se il motore non risponde al comando
4	CRI/DEG	Fault della ruota front right: degradato se encoder non fornisce feedback, critico se il motore non risponde al comando
5	CRI/DEG	Fault della ruota rear left: degradato se encoder non fornisce feedback, critico se il motore non risponde al comando
6	CRI/DEG	Fault della ruota rear right: degradato se encoder non fornisce feedback, critico se il motore non risponde al comando
7	CRI/DEG	Supervisore della Board 2 aggiornato lentamente o non aggiornato

Table 4: Mappatura delle fault mask generate dalla Board 1

8.4 Rilevazione Faults locali

La rilevazione dei faults locali è implementata dal Chart **Monitor Board Local Status**. Questo blocco riceve in ingresso gli snapshot aggiornati delle diverse periferiche e produce in uscita lo stato di ciascun componente monitorato.

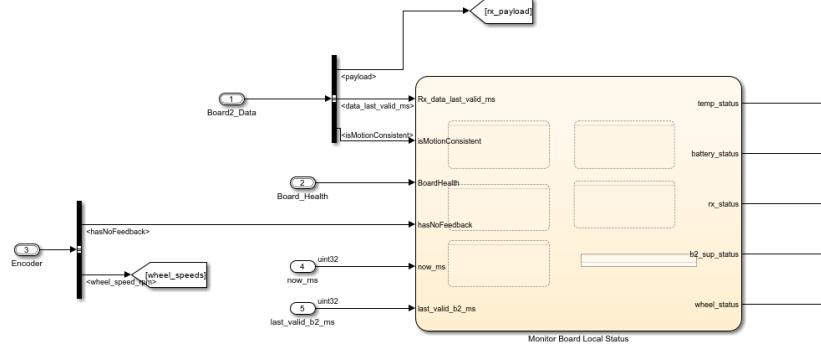


Figure 46: Chart per rilevazione fault locali

Il sottosistema è composto da più sottoblocchi, ciascuno dedicato al monitoraggio di una specifica periferica o funzionalità:

- **MonitorTemperature**: verifica che la temperatura sia all'interno del range operativo e che le misure siano aggiornate correttamente;
- **MonitorBattery**: verifica che il livello della batteria sia compatibile con il funzionamento del sistema;
- **MonitorRx**: verifica la correttezza e la frequenza di ricezione dei messaggi dalla Board 2;
- **MonitorBoard2Supervisor**: verifica che il supervisore della Board 2 sia attivo e aggiornato correttamente;
- **MonitorWheels**: verifica lo stato delle singole ruote, utilizzando il feedback degli encoder e le informazioni di coerenza del moto.

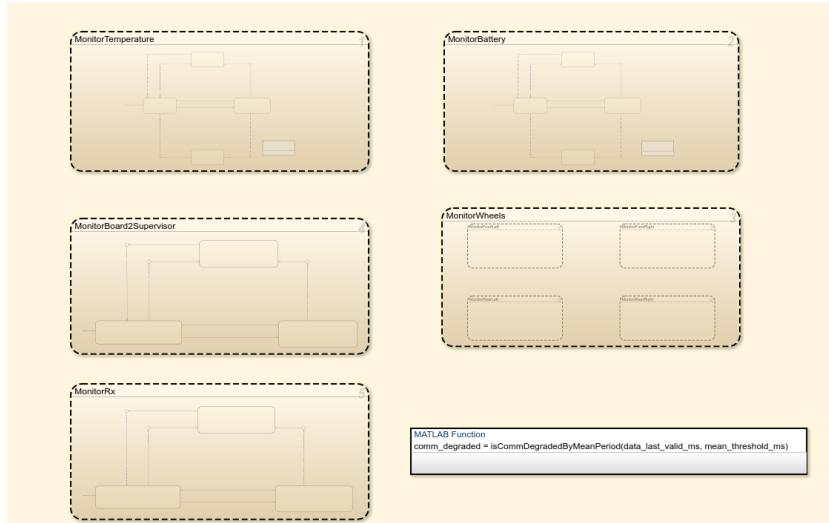


Figure 47: Stati paralleli del chart Monitor Board Local Status.

Ciascun sottoblocco implementa uno Stateflow chart che classifica lo stato della periferica associata in tre possibili condizioni:

- **OK**: funzionamento nominale;
- **DEGRADED**: funzionamento degradato, ma ancora operativo;
- **CRITICAL**: fault critico che compromette il funzionamento corretto o sicuro.

Questi sottoblocchi operano in modo indipendente e in parallelo, permettendo al supervisore di valutare contemporaneamente lo stato di tutte le periferiche monitorate.

8.4.1 Esempio: monitoraggio della ricezione dalla Board 2

Il sottoblocco *MonitorRx*, ha il compito di verificare la frequenza di ricezione di messaggi validi provenienti dalla Board 2.

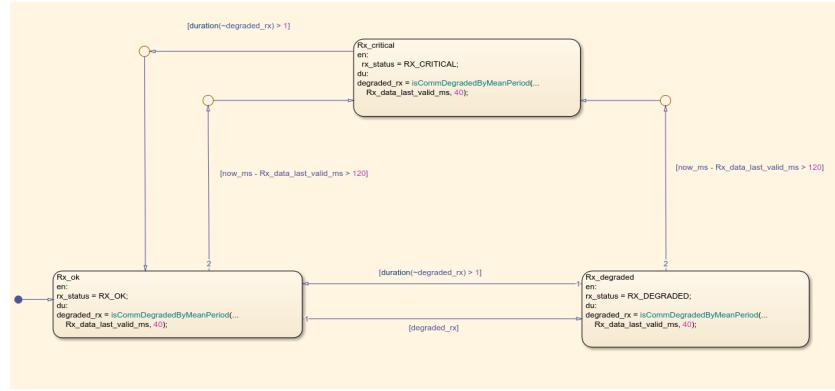


Figure 48

Il blocco implementa uno Stateflow chart con tre possibili stati: **RX_OK**, **RX_DEGRADED** e **RX_CRITICAL**. La transizione tra questi stati avviene in funzione del tempo dell'ultimo aggiornamento valido ricevuto (*Rx_data_last_valid_ms*) e del tempo corrente (*now_ms*).

In particolare:

- il sistema si trova nello stato **RX_OK** quando i messaggi vengono ricevuti con la frequenza attesa;
- lo stato passa a **RX_DEGRADED** quando la frequenza di ricezione risulta inferiore ad una certa soglia (in questo caso 1 messaggio ogni 40ms). Questa condizione viene rilevata tramite la funzione: `isCommDegradedByMeanPeriod(Rx_data_last_valid_ms, threshold)` che verifica se il periodo medio di aggiornamento supera la soglia prevista;
- lo stato passa a **RX_CRITICAL** quando non vengono ricevuti aggiornamenti per un intervallo di tempo superiore alla soglia di timeout, ovvero quando: `now_ms - Rx_data_last_valid_ms > TIMEOUT`

Quando la comunicazione riprende con una frequenza adeguata, il sistema ritorna automaticamente allo stato **RX_OK**.

Questa logica consente di distinguere tra una comunicazione semplicemente rallentata (stato degradato) e una completa perdita della comunicazione (stato critico), permettendo al supervisore di reagire in modo appropriato nelle successive fasi di decisione e attuazione, nonché di ritornare automaticamente a uno stato nominale quando la comunicazione viene ripristinata.

8.4.2 Output del monitoraggio

Gli stati calcolati dai diversi Subcharts vengono forniti in uscita come segnali distinti che verranno successivamente utilizzati per la costruzione delle fault mask e la determinazione dello stato globale del rover.

temp_status. Il segnale *temp_status* indica lo stato della temperatura della board e può assumere i seguenti valori:

- TEMP_HEALTH_OK: quando la temperatura è nel range $]-5; 55[$;
- TEMP_HEALTH_DEGRADED: quando la temperatura è nel range $]-15; -5] \cup [55; 60[$;
- TEMP_HEALTH_CRITICAL:
 - quando la temperatura permane per almeno 4 s nel range $]-\infty; -15] \cup [60; +\infty[$;
 - oppure quando non vengono ricevuti aggiornamenti per almeno 0.5 s e la temperatura stimata, assumendo nel caso peggiore un aumento di $1^{\circ}\text{C}/\text{s}$, supera i 65°C .

batt_status. Il segnale *batt_status* indica lo stato della batteria e può assumere i seguenti valori:

- BATT_HEALTH_OK: quando la percentuale di batteria è maggiore del 25%;
- BATT_HEALTH_DEGRADED: quando la percentuale di batteria è minore del 23%;
- BATT_HEALTH_CRITICAL:
 - quando la percentuale permane per almeno 5 s nel range $[0\%; 15\%]$;
 - oppure quando non vengono ricevuti aggiornamenti per almeno 0.5 s e la percentuale stimata, assumendo una diminuzione nel caso peggiore di $0.42\%/\text{s}$, scende sotto il 15%.

wheel_status. Il segnale *wheel_status(i)* indica lo stato della ruota i e può assumere i seguenti valori:

- WHEEL_OK: rappresenta la condizione di funzionamento nominale. In questo stato, l'encoder della ruota fornisce feedback valido (ovvero $has_no_feedback(i)$ è falso) e il moto del rover risulta coerente, come verificato dal segnale $isMotionConsistent$;
- WHEEL_DEGRADED_ENCODER: quando il segnale booleano $has_no_feedback(i)$, fornito dallo snapshot degli encoder, indica assenza di rotazione misurata per la ruota i nonostante il motore sia attivo. Questa condizione indica una perdita del feedback dell'encoder, ma il sistema può continuare a operare utilizzando una logica di fallback che stima la velocità della ruota a partire dalle altre ruote funzionanti;
- WHEEL_CRITICAL_MOTOR: quando alla condizione $has_no_feedback(i)$ si aggiunge un'incoerenza globale del moto, rilevata tramite il segnale $isMotionConsistent$. Questa combinazione indica che l'assenza di feedback non è dovuta a un guasto dell'encoder, ma al fatto che il motore della ruota i non risponde correttamente al comando.

b2_sup_status. Il segnale $b2_sup_status$ indica lo stato del supervisore della Board 2 e può assumere i seguenti valori:

- SUP_OK: quando il supervisore della Board 2 risulta aggiornato regolarmente;
- SUP_DEGRADED: quando il periodo medio di aggiornamento del supervisore della Board 2 supera il valore nominale, indicando un rallentamento nell'esecuzione;
- SUP_CRITICAL: quando l'intervallo di tempo dall'ultimo aggiornamento valido supera i 120 ms, indicando che il supervisore della Board 2 non è più attivo o non comunica correttamente.

rx_status. Il segnale rx_status indica lo stato della comunicazione con la Board 2 e può assumere i seguenti valori:

- RX_OK: quando i messaggi vengono ricevuti con la frequenza prevista e la variabile $data_last_valid_ms$ viene aggiornata regolarmente dal task di ricezione;
- RX_DEGRADED: quando la comunicazione risulta instabile e la frequenza media di ricezione dei messaggi validi diminuisce. Questa condizione viene rilevata monitorando la media degli intervalli tra aggiornamenti successivi e verificando se essa supera la soglia di 40 ms;

- RX_CRITICAL: quando l’intervallo di tempo dall’ultimo aggiornamento valido supera i 120 ms, indicando una perdita della comunicazione con la Board 2.

Costruzione delle maschere critiche e degradate A partire dagli stati critici e degradati rilevati durante la fase di monitoraggio, vengono costruite due maschere di errore (*bitmask*), una per le anomalie critiche e una per quelle degradate.

Questa rappresentazione è stata preferita ad altre perché consente una gestione efficiente dei fault lato firmware e permette di trasmettere facilmente lo stato della Board 1 alla Board 2 tramite il canale di comunicazione.

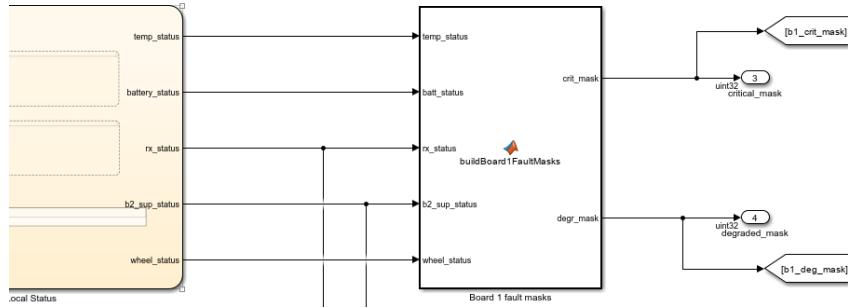


Figure 49: Costruzione delle maschere di errore critiche e degradate nel supervisore.

Il significato dei singoli bit è descritto nella Tabella 4.

8.5 Selezione della board attuante

Il chart *Decide actuation privileges* mostrato in figura decide quale board è responsabile dell’attuazione.

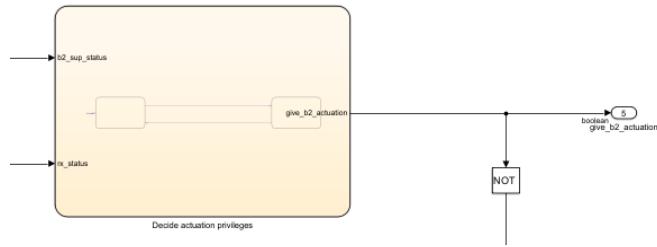


Figure 50: Stateflow chart per la selezione della board attuante.

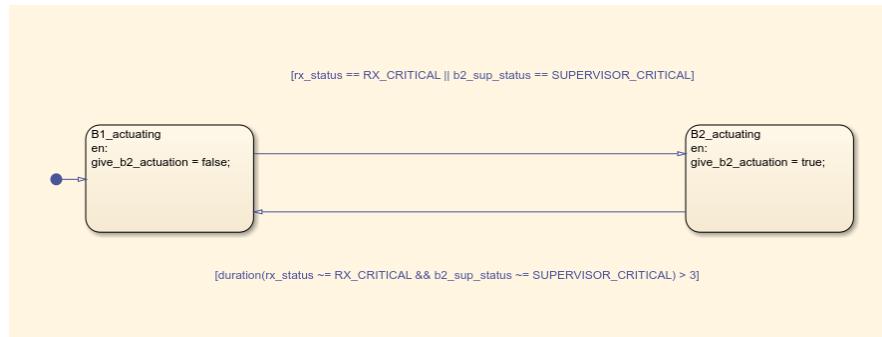
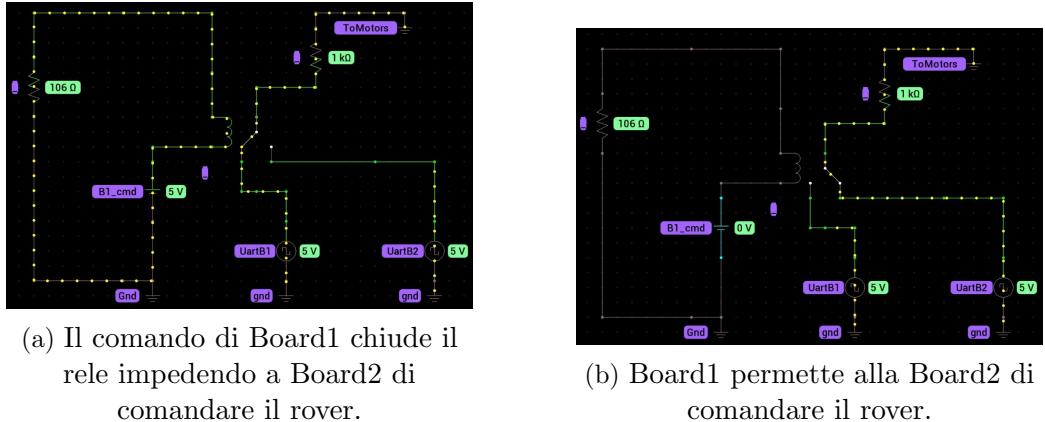


Figure 51: Stati interni del chart Decide actuation privileges.

La selezione è regolata dalla variabile *give_b2_actuation*, secondo la seguente logica:

- *give_b2_actuation = 1*: il controllo dell'attuazione viene passato alla Board 2. Questa condizione si verifica quando viene rilevata una condizione critica nella comunicazione (*rx_status = RX_CRITICAL*) oppure nello stato del supervisore della Board 2 (*b2_sup_status = SUP_CRITICAL*).
- *give_b2_actuation = 0*: il controllo dell'attuazione rimane alla Board 1. Questa condizione viene mantenuta finché non sono presenti condizioni critiche; inoltre, se il controllo è stato precedentemente ceduto alla Board 2, la Board 1 riprende l'attuazione solo quando entrambe le condizioni critiche non sono più presenti e tale situazione permane stabilmente per almeno 3 s.

Tale meccanismo agisce come un dispositivo di sicurezza hardware basato sullo stato del supervisore, come mostrato nelle Figure 52b e 52a.



8.6 Stato del rover e calcolo dei riferimenti (solo quando attua Board 1)

Quando la Board 1 è responsabile dell’attuazione (ovvero $give_b2_actuation = 0$), viene abilitato il sottosistema *Rover state and Actuation decisions*.

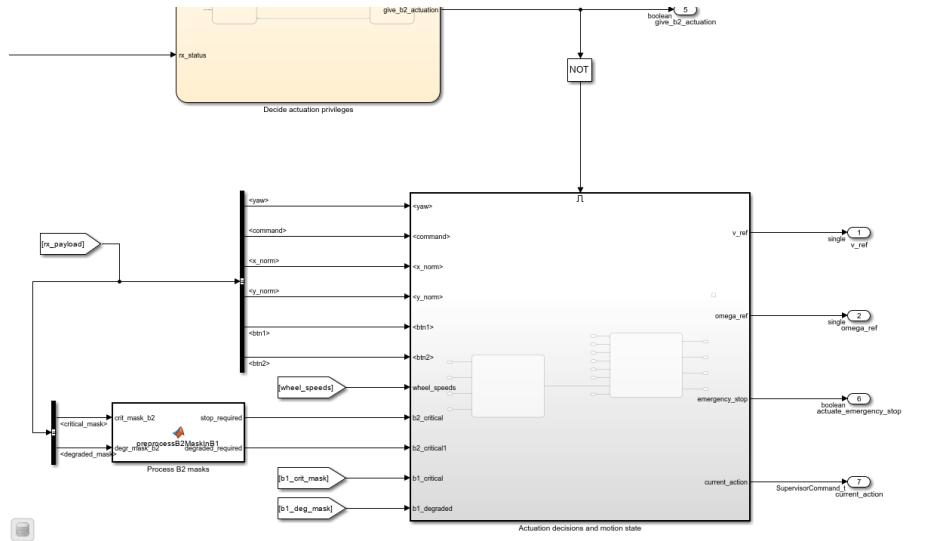


Figure 53: Enabled subsystem Rover state and Actuation decisions

Questo sottosistema ha il compito di ricostruire lo stato operativo globale del rover a partire dai fault locali e remoti e di produrre in uscita i riferimenti v_ref e ω_{ref} , oltre ai segnali $emergency_stop$ e $current_action$.

Pre-processamento delle fault mask di Board 2 Le fault mask ricevute dalla Board 2 prima di essere usate vengono pre-processate tramite la funzione `preprocessB2MaskInB1`. L'obiettivo è distinguere i fault remoti che richiedono un arresto immediato del rover da quelli che, pur essendo critici, possono essere gestiti senza emergency stop (ad esempio fault legati alla comunicazione).

In particolare, viene generato:

- *stop_required*: viene posto a 1 quando la Board 2 segnala fault critici su sottosistemi essenziali per il controllo del rover (ESP o IMU), senza i quali non è possibile garantire un funzionamento sicuro;
- *degraded_required*: viene posto a 1 se è presente almeno un fault degradato remoto, oppure se la Board 2 segnala fault critici legati a ricezione o lettura dei dati prodotti dal supervisore di Board 1 (fault RX o SUP).

Questa distinzione permette alla Board 1 di continuare ad attuare in sicurezza anche quando la Board 2 presenta problemi di ricezione da Board 1, attivando in tal caso la modalità degradata invece dell'arresto di emergenza.

8.6.1 Struttura interna del sottosistema di decisione

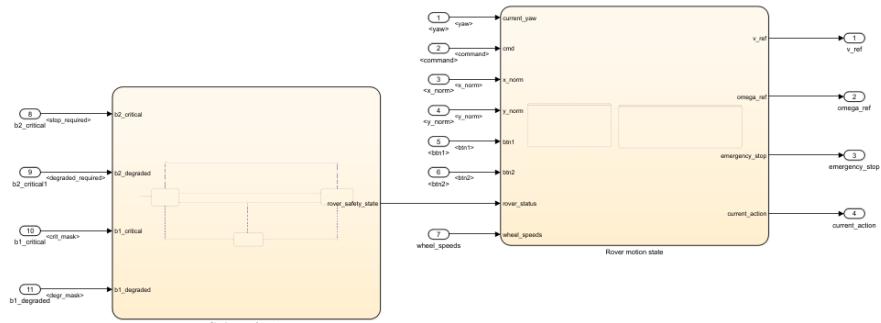


Figure 54: Struttura interna del sottosistema Rover state and Actuation decisions

Il sottosistema è composto da due Stateflow chart principali, organizzati in cascata. Il primo chart, denominato *Check rover safety state*, ha il compito di integrare le informazioni di fault locali e remoti per determinare lo stato globale di sicurezza del rover. Il secondo chart, *Rover motion state*, utilizza questo stato globale insieme ai comandi ricevuti per determinare l'azione corrente del rover e calcolare i riferimenti di velocità.

Check rover safety state Il chart *Check rover safety state* determina lo stato di sicurezza globale del rover (*rover_safety_state*) combinando le condizioni di fault rilevate dalla Board 1 e dalla Board 2.

In ingresso riceve quattro segnali booleani: *b1_critical* e *b1_degraded*, derivati dalle fault mask locali, e *b2_critical* e *b2_degraded*, ottenuti dal pre-processamento delle mask ricevute dalla Board 2.

Il chart può assumere tre stati mutuamente esclusivi:

- *SAFETY_OK*: attivo quando non sono presenti fault critici né degradati;
- *SAFETY_DEGRADED*: attivo quando è presente almeno una condizione degradata su una delle due board, in assenza di fault critici;
- *SAFETY_CRITICAL*: attivo quando è presente almeno un fault critico su una delle due board.

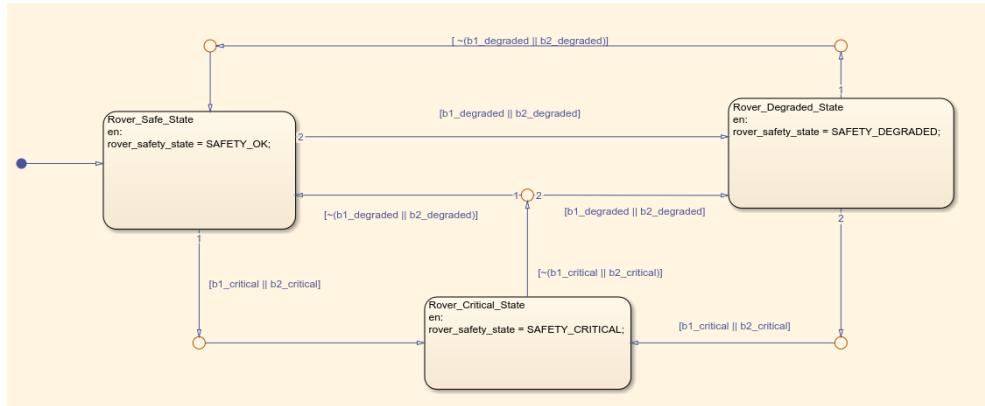


Figure 55: Struttura del chart Check rover safety state

Rover motion state Il chart *Rover motion state* ha il compito di determinare l'azione corrente del rover e calcolare i riferimenti di velocità *v_ref* e *omega_ref*, a partire dai comandi ricevuti e dallo stato di sicurezza globale *rover_safety_state* calcolato dal blocco precedente.

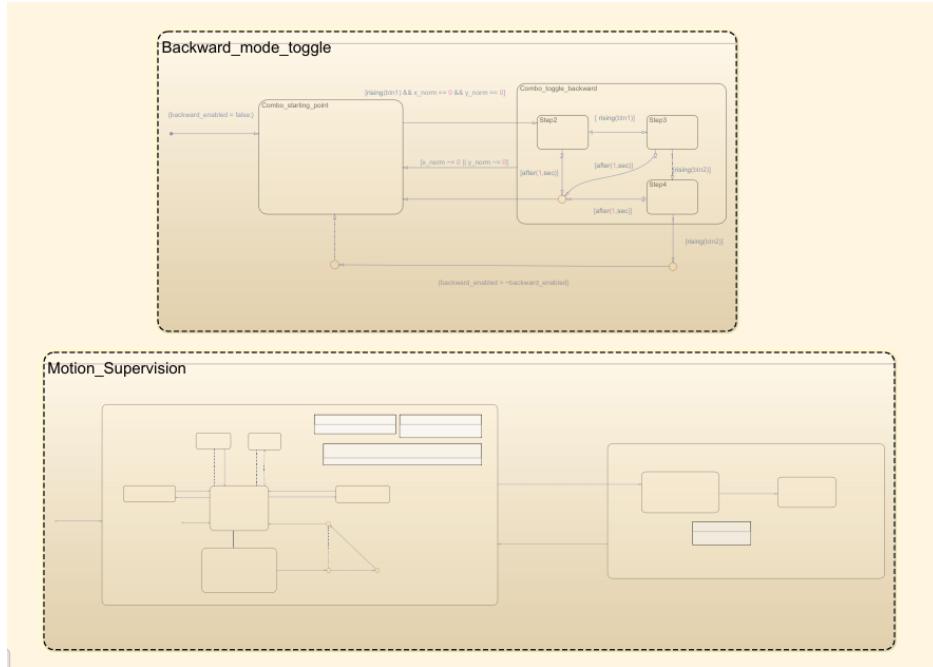


Figure 56: Struttura del chart Rover motion state.

Il chart è composto da due stati paralleli. Il primo stato, denominato *Backward_mode_toggle*, gestisce la commutazione tra retromarcia normale e rotazione a 180 gradi quando il comando utente richiede un movimento all’indietro. Il secondo stato, *Motion_Supervision*, implementa la macchina a stati principale che determina il tipo di movimento del rover e calcola i riferimenti di velocità.

Backward_mode_toggle Lo stato *Backward_mode_toggle* gestisce la variabile interna *backward_enabled*, che determina il comportamento del rover quando il comando di velocità longitudinale è negativo.

In particolare, quando l’utente spinge il joystick all’indietro ($y_norm < 0$), il rover può operare in due modalità:

- *retromarcia*, in cui il rover si muove direttamente all’indietro;
- *rotazione a 180 gradi*, in cui il rover ruota sul posto fino a invertire la propria direzione e successivamente riprende il moto in avanti.

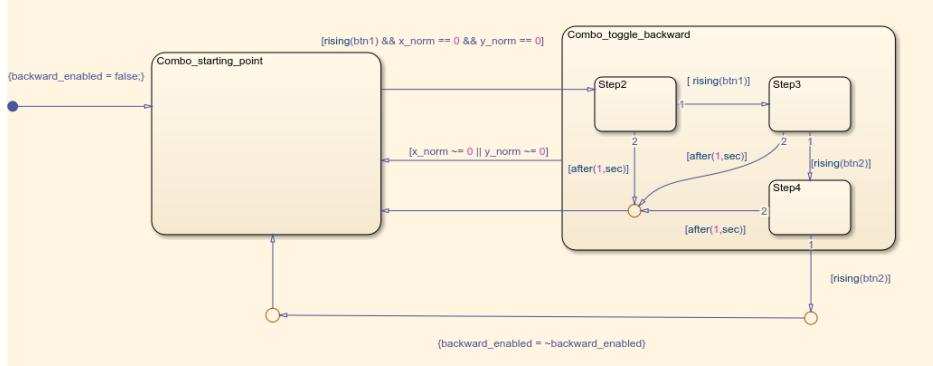


Figure 57: Struttura interna del subchart Backward mode toggle

La selezione tra queste due modalità avviene tramite una combinazione specifica di input (gestita dal chart), che consente di abilitare o disabilitare la retromarcia diretta. In questo modo si evita che il rover entri accidentalmente in retromarcia, permettendo invece una manovra controllata di inversione di direzione quando necessario.

La variabile *backward_enabled* viene quindi utilizzata dal blocco *Motion_Supervision* per determinare se interpretare un comando negativo come retromarcia oppure come richiesta di rotazione a 180 gradi.

Rover motion state Il chart *Rover motion state* implementa la logica principale di movimento del rover. Esso è organizzato in due macro-stati mutuamente esclusivi: uno dedicato alla marcia normale e uno dedicato alla gestione dell'arresto di emergenza. La transizione tra questi stati dipende sia dai comandi ricevuti dalla Board 2 sia dallo stato di sicurezza globale del rover.

Stato di marcia normale Lo stato *Stato_marcia_rover* rappresenta il funzionamento nominale del rover e contiene al suo interno i diversi stati operativi associati al movimento, tra cui:

- marcia normale, in cui i riferimenti sono direttamente derivati dall'input utente (x_norm , y_norm);
- manovre automatiche, come schivata verso sinistra o destra;
- rotazione a 180 gradi, utilizzata per invertire la direzione del rover;
- altre manovre controllate, in funzione del comando di sicurezza ricevuto da Board 2 *command*.

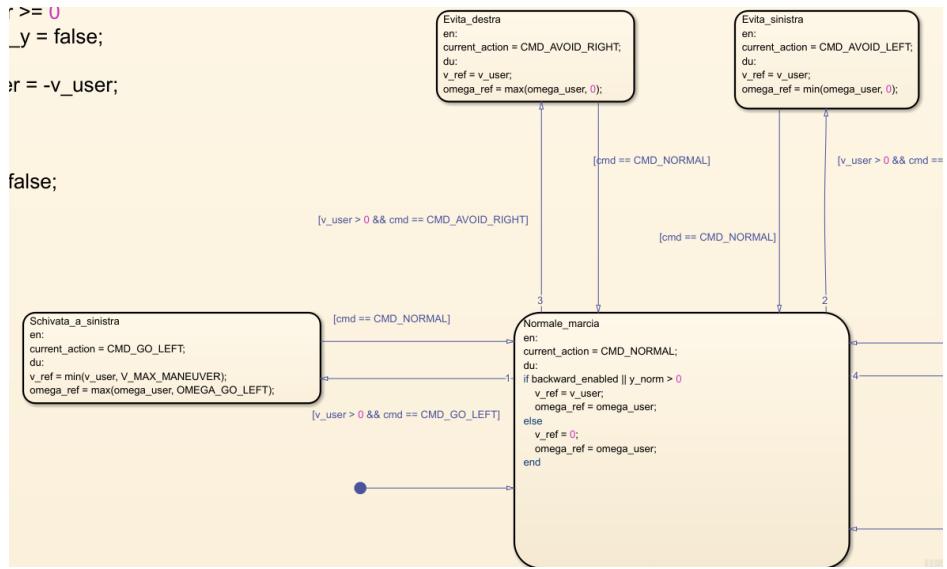


Figure 58: Esempio dei sotto-stati e transizioni che compongono lo stato di marcia rover

All'interno di questo stato, i riferimenti v_ref e ω_ref vengono calcolati a partire dagli ingressi utente e limitati in base allo stato di sicurezza $rover_status$.

La transizione fuori da questo stato avviene quando viene richiesto un arresto di emergenza, che può essere causato da:

- la ricezione del comando CMD_ESTOP dalla Board 2 (a seguito del rilevamento di un ostacolo tramite sonar) mentre il rover sta procedendo in avanti;
- il passaggio dello stato globale del rover a $SAFETY_CRITICAL$.

Stato di arresto di emergenza Quando viene attivata una condizione di arresto di emergenza, il chart entra nello stato $Procedura_emergency_stop$. In questo stato:

- i riferimenti v_ref e ω_ref vengono forzati a zero;
- il segnale $emergency_stop$ viene posto a vero;
- viene monitorata la velocità delle ruote per verificare l'effettivo arresto del rover.

Una volta rilevato che le ruote sono ferme, il sistema rimane nello stato di frenata di emergenza finché la condizione di arresto, verificata tramite gli

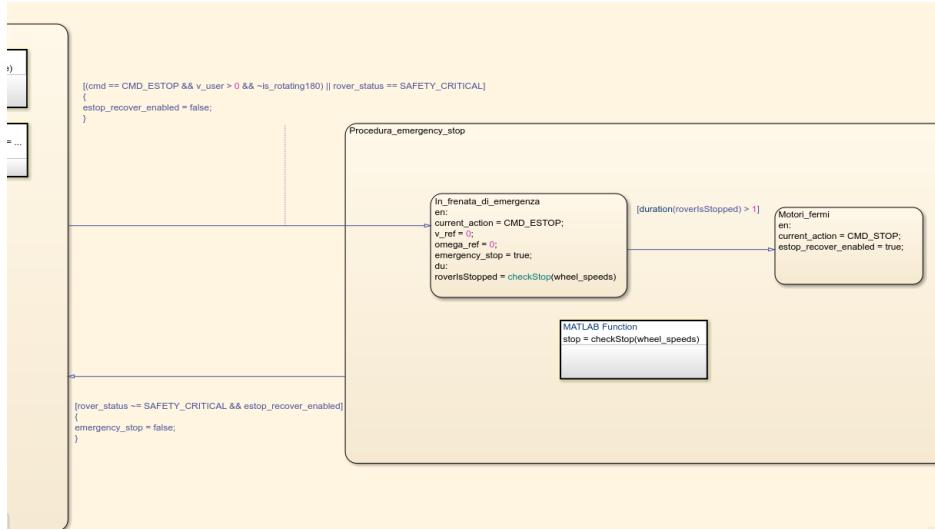


Figure 59: Stato frenata di emergenza e transizioni di attivazione (provenienti dallo stato normale marcia)

encoder, permane per almeno 1s consecutivo. Questa verifica temporale garantisce che l’arresto sia effettivo e stabile. Solo quando tale condizione è soddisfatta, il sistema transita nello stato *Motori_fermi*, che rappresenta la condizione di arresto completo del rover.

Una volta raggiunto lo stato *Motori_fermi*, il sistema può uscire da questa condizione non appena non sono più presenti fault critici. In questa situazione, il rover ritorna nello stato di marcia normale, anche se il comando *CMD_ESTOP* è ancora attivo, poiché il rover è già fermo e tale condizione non rappresenta un rischio immediato.

Questo consente al rover di eseguire manovre sicure come rotazioni sul posto o retromarcia. Tuttavia, se il comando *CMD_ESTOP* persiste e viene richiesto un movimento in avanti, il sistema transita nuovamente nello stato di frenata di emergenza, impedendo l’avanzamento verso l’ostacolo.

9 Supervisore Board 2

9.1 Panoramica generale

Il supervisore della Board 2 è implementato come un modulo Simulink denominato **SupervisorB2**, il cui schema a blocchi è illustrato in Figura 60.

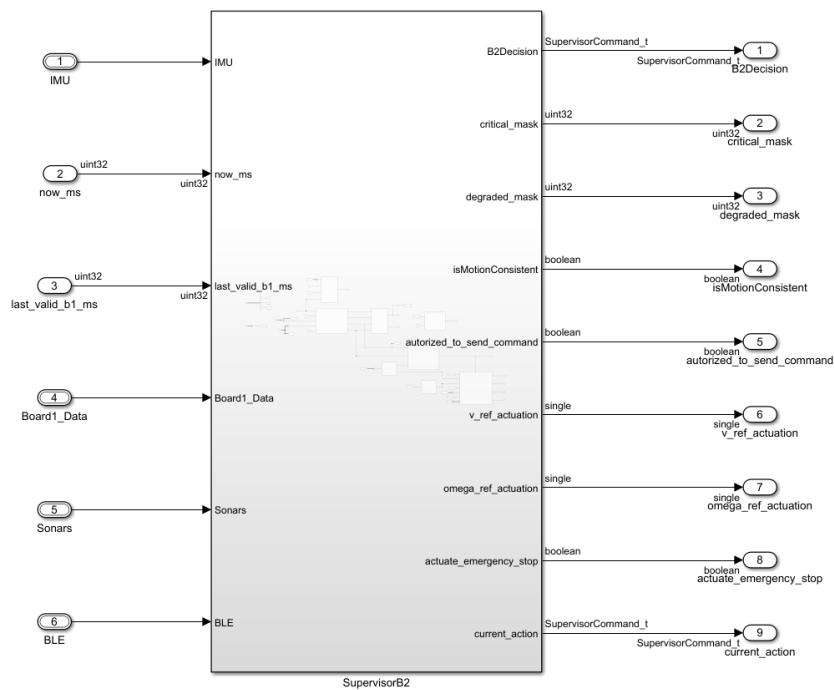


Figure 60: Schema a blocchi del modulo SupervisorB2.

Il modulo determina la direzione del rover (*B2Decision*) e verifica la coerenza tra la stima odometrica (velocità ruote) e i dati dell'IMU. Inoltre, qualora autorizzato o in caso di mancata comunicazione con la Board 1, calcola i riferimenti di velocità lineare e angolare.

L'architettura è composta da 5 parti principali:

- **Rilevamento anomalie locali:** monitora lo stato di comunicazione con ESP32, IMU e Board 1, classificandola come OK, DEGRADED o CRITICAL.
- **Generazione maschere di errore:** aggrega le anomalie rilevate nelle maschere locali (*fault masks*), per rappresentare lo stato di salute della Board 2.
- **Logica che abilita l'attuazione:** autorizza la Board 2 all'attuazione

in caso di interruzione della comunicazione con la Board 1 o su richiesta esplicita di quest'ultima.

- **Gestione ostacoli:** analizza i dati dei sensori sonar per correggere il comando di movimento in presenza di ostacoli statici o in movimento.
- **Calcolo riferimenti e stato globale:** ricostruisce lo stato del sistema combinando le maschere critiche delle due board. Se abilitata all'attuazione, la Board 2 genera i riferimenti di velocità dimezzandone i valori nominali per operare in modalità degradata.

Nel seguito verranno descritti i segnali di input e output del supervisore, successivamente verranno descritte le quattro parti principali del supervisore descritte sopra.

9.2 Ingressi del supervisore

I segnali in input che riceve sono:

- **Board1_Data:** è l'ultimo snapshot di ricezione (*RxSnapshot*) aggiornato dal task di ricezione di Board2. I segnali utilizzati dal supervisore sono:
 - *wheel_speed_rpm(i)*: è la velocità in RPM di ogni ruota.
 - *critical_mask*: è una maschera di errore a 8 bit calcolata da Board 1, in cui ogni bit indica la presenza di un'anomalia *critica* specifica.
 - *data_last_valid_ms*: rappresenta il tempo in millisecondi dell'ultimo dato valido ricevuto da Board 1, utilizzato per rilevare eventuali timeout nella ricezione.
- **IMU:** è l'ultimo snapshot (*IMUSnapshot*) di rilevamento dell'angolo yaw. La struttura dati è la seguente:

```

>typedef struct
{
    uint32_t task_last_run_ms;      /* ultima esecuzione del task */
    uint32_t data_last_valid_ms;   /* ultimo istante in cui i dati sono validi */

    /* Accelerometer (g) */
    float ax_g;
    float ay_g;
    float az_g;

    /* Gyroscope (deg/s) */
    float gx_dps;
    float gy_dps;
    float gz_dps;

    /* Temperature (°C) */
    float temperature_degC;

    float yaw;

} IMUSnapshot_t;
```

Figure 61: Struttura dati IMU_snapshot.

- **Sonars:** è l'ultimo snapshot delle distanze dagli ostacoli rilevati. La struttura dati è la seguente:

```

typedef struct
{
    uint16_t dist_cm[3];

    uint32_t task_last_run_ms;      /* ultima esecuzione del task */
    uint32_t data_last_valid_ms[3]; /* ultimo istante in cui i dati sono validi */
} SonarSnapshot_t;
```

Figure 62: Struttura dati sonar_snapshot.

- **now_ms:** rappresenta il tempo corrente in millisecondi, utilizzato per la gestione delle temporizzazioni e il rilevamento di timeout.
- **last_valid_b1_ms:** rappresenta il tempo, in millisecondi, dell'ultimo istante in cui la comunicazione con la Board 1 è stata considerata valida. Questo valore viene aggiornato quando vengono ricevuti nuovi dati coerenti dalla Board 2 ed è utilizzato dal supervisore per rilevare eventuali fault o relativi alla comunicazione.

9.3 Uscite del supervisore

I segnali in output che fornisce sono:

- **B2Decision:** è il comando generato dal supervisore della Board 2 in funzione degli ostacoli rilevati. I valori che può assumere sono:

```

typedef enum
{
    CMD_NORMAL = 0,
    CMD_ROTATE_180,
    CMD_GO_LEFT,
    CMD_GO_RIGHT,
    CMD_AVOID_RIGHT,
    CMD_AVOID_LEFT,
    CMD_STOP,
    CMD_ESTOP
} SupervisorCommand_t;

```

Figure 63: Comandi in uscita dal supervisore della Board 2.

- **critical_mask & degraded_mask**: sono due mask di errore che rappresentano le anomalie critiche e degradate rilevate dalla Board 2.
- **isMotionConsistent**: rappresenta un segnale booleano che indica se c’è coerenza tra la rotazione misurata dalla IMU e quella stimata dalla velocità dei motori.
- **authorized_to_send_command**: rappresenta un segnale booleano che indica se la Board2 può inviare comandi di movimento al rele collegato ai driver motori.
- **v_ref_actuation**: rappresenta il riferimento di velocità lineare del rover, in RPM, calcolato dal supervisore di Board 2 in base ai comandi ricevuti dall’utente. Tale riferimento viene attuato dalla Board 2 solo se il segnale *authorized_to_send_command* è attivo.
- **omega_ref_actuation**: rappresenta il riferimento di velocità angolare del rover, in rad/s, calcolato dal supervisore di Board 2 in base ai comandi ricevuti dall’utente. Tale riferimento viene attuato dalla Board 2 solo se il segnale *authorized_to_send_command* è attivo.
- **actuate_emergency_stop**: rappresenta un segnale booleano che indica se il rover deve essere fermato immediatamente. Tale segnale viene attivato solo se il segnale *authorized_to_send_command* è attivo.
- **current_action**: se Board 2 è abilitata ad attuare, rappresenta un segnale che indica l’azione attualmente intrapresa dal supervisore.

La corrispondenza tra le fault mask e le anomalie rilevate è riportata nella Tabella 5.

Bit	Mask	Descrizione dell'Anomalia
0	CRI/DEG	Timeout della comunicazione con ESP32
1	CRI/DEG	Timeout della comunicazione con il sensore IMU
2	CRI/DEG	Comunicazione con Board 1 lenta o in timeout
3	CRI/DEG	Supervisore Board 1 aggiornato lentamente o non aggiornato
4-7	Reserved	Bit riservati per espansioni

Table 5: Mappatura della maschera di errore della Board 2

9.4 Rilevazione Faults

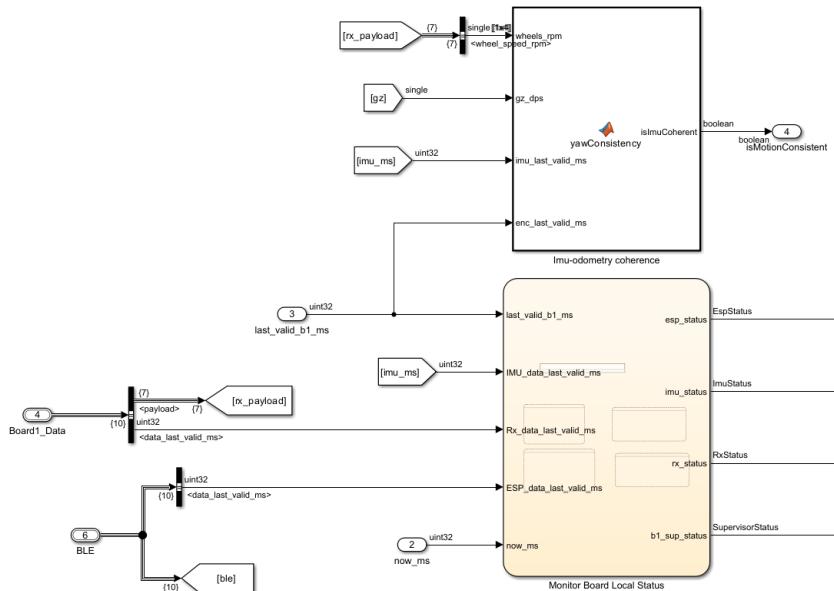


Figure 64: Chart di gestione dei faults.

Analogamente al supervisore di Board 1, la rilevazione dei faults locali è implementata dal Chart **Monitor Board Local Status**. Questo blocco riceve in ingresso gli snapshot aggiornati delle diverse periferiche e produce in uscita lo stato di ciascun componente monitorato.

Il sottosistema è composto da più sottoblocki, ciascuno dedicato al monitoraggio di una specifica periferica o funzionalità:

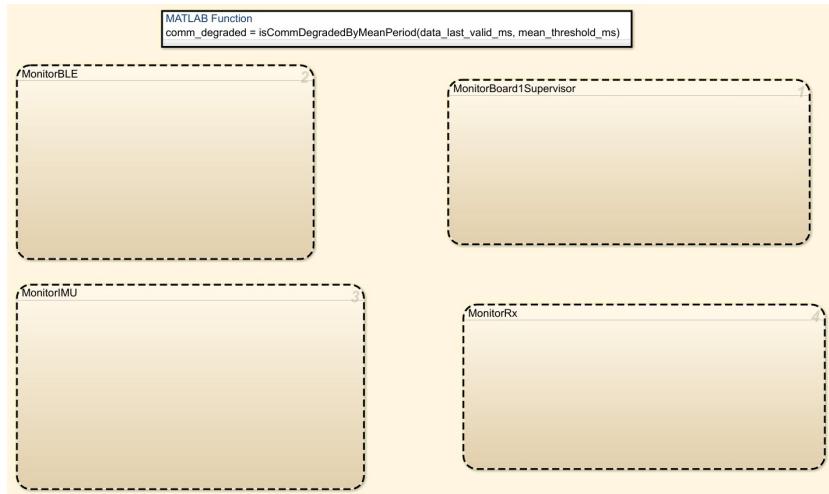


Figure 65: Sottoblocchi che si occupano di rilevare anomalie nelle periferiche monitorate da Board 2.

Questi sottoblocchi operano in modo indipendente e in parallelo, permettendo al supervisore di valutare contemporaneamente lo stato di tutte le periferiche monitorate.

MonitorBLE Monitora e rileva un ritardo eccessivo nell'aggiornamento dei dati ricevuti dall'ESP32. L'output associato a questo sottosistema è **esp_status** e può assumere i seguenti valori:

- **ESP_DEGRADED** : se la media mobile degli intervalli di ricezione dati dall'ESP32 supera la soglia di 40 ms.
- **ESP_CRITICAL**: se la media mobile degli intervalli di ricezione dati dall'ESP32 supera la soglia di 120 ms.
- **ESP_OK** : quando la frequenza media di aggiornamento rientra nei limiti nominali di sicurezza stabiliti.

MonitorBoard1Supervisor Verifica che il supervisore della Board 2 sia attivo e aggiornato correttamente. L'output associato a questo sottosistema è **b1_sup_status** e può assumere i seguenti valori:

- **SUP_DEGRADED** : questo valore indica una discontinuità operativa della Board 1. Il supervisore della Board 2 monitora l'heartbeat (una variabile) del supervisore remoto, il quale incrementa il valore ogni volta

che viene eseguito. Se la media degli ultimi 10 intervalli di aggiornamento superi i 40 ms, il sistema segnala uno stato di degrado del supervisore di Board 1.

- **SUP_CRITICAL** : se l’intervallo di tempo dall’ultimo aggiornamento dell’heartbeat del supervisore di Board1 supera i 120 ms, imposta questo valore.
- **SUP_OK** : quando non ci sono ne condizioni critiche ne degradate imposta questo valore.

MonitorIMU Monitora e rileva un ritardo eccessivo nell’aggiornamento dei dati ricevuti dall’IMU. L’output associato a questo sottosistema è **imu_status** e può assumere i seguenti valori:

- **IMU_DEGRADED** : se la media mobile degli intervalli di aggiornamento dell’angolo di yaw dell’IMU supera i 40 ms.
- **IMU_CRITICAL** : se la media mobile degli intervalli di aggiornamento dell’angolo di yaw dell’IMU supera i 120 ms.
- **IMU_OK** : quando la frequenza media dei dati inerziali rientra nei parametri nominali di funzionamento.

MonitorRx Rileva un ritardo eccessivo nell’aggiornamento dei dati ricevuti dalla Board 1. L’output associato a questo sottosistema è **rx_status** e può assumere i seguenti valori:

- **RX_DEGRADED** : questo valore identifica una comunicazione instabile. Sebbene il collegamento fisico sia attivo, fattori quali errori di checksum (CRC) o anomalie nella lunghezza dei pacchetti impediscono l’aggiornamento della variabile `data_last_valid_ms`. Il sistema monitora la qualità del link calcolando la media mobile degli ultimi 10 intervalli di ricezione valida; se tale media supera la soglia critica di 40 ms, viene segnalato il degrado della ricezione.
- **RX_CRITICAL** : se l’intervallo di tempo dall’ultima ricezione corretta supera i 120 ms, imposta questo valore.
- **RX_OK** : quando non ci sono ne condizioni critiche ne degradate imposta questo valore.

In aggiunta ai fault descritti, il supervisore 2 monitora anche la coerenza del movimento del rover data l’azione di movimento intrapresa dal rover.

In particolare, la funzione Matlab *Imu-odometry coherence* monitora tale coerenza prendendo in ingresso i valori del sensore IMU e la velocità delle ruote proveniente dallo snapshot di ricezione *RxSnapshot* e dando in uscita la variabile booleana *isMotionConsistent* che può assumere i seguenti valori:

- **0**: indica che c'è una discrepanza significativa tra la rotazione misurata dalla IMU e quella stimata dalla velocità dei motori, suggerendo un possibile guasto o malfunzionamento;
- **1**: indica che i dati della IMU e le stime basate sulla velocità dei motori sono coerenti, suggerendo un funzionamento normale del sistema di movimento.

In condizioni di funzionamento nominali, questa variabile verrà utilizzata da Board 1 per rilevare un guasto critico ad uno dei motori.

9.5 Costruzione maschere degradate e critiche

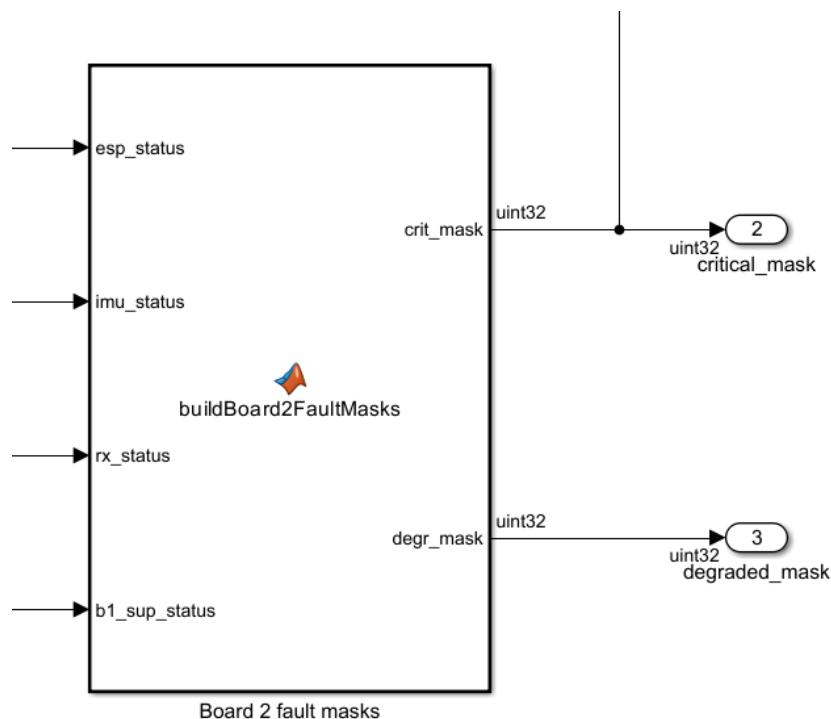


Figure 66: Chart per l'aggregazione delle faults.

Dagli stati critici e degradati rilevati nella parte di gestione faults, si costruiscono due maschere di errore a 8 bit, una per le anomalie critiche e una per quelle degradate.

Il bit di ogni maschera rappresenta un'anomalia specifica, come descritto nella Tabella 6.

Bit	Componente	critical_mask	degraded_mask
0	ESP32	1 se ESP_CRITICAL	1 se ESP_DEGRADED
1	IMU	1 se IMU_CRITICAL	1 se IMU_DEGRADED
2	Ricezione da Board 1	1 se RX_CRITICAL	1 se RX_DEGRADED
3	Supervisore Board 1	1 se SUP_CRITICAL	1 se SUP_DEGRADED
4-7	Reserved	Bit non usati	Bit non usati

Table 6: Mappatura dei Bit nelle Fault Masks di Board 2

9.6 Logica che consente a Board 2 di attuare

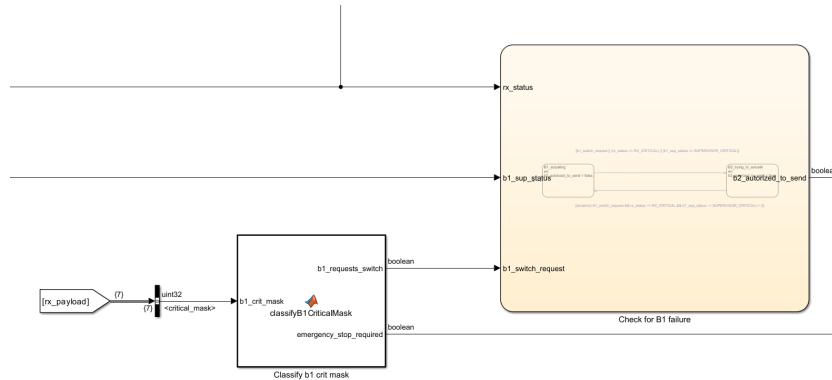


Figure 67: Chart per decidere se autorizzare o meno la Board2 a muovere il rover.

Board 2 può trovarsi nelle condizioni di poter attuare il comando ai motori. Queste condizioni abilitano una variabile booleana *authorized_to_send_command* che assume valore positivo o negativo nei seguenti casi:

- **authorized_to_send_command = 1**: Board 2 è autorizzata a inviare comandi di movimento al rover. Questa condizione si attiva quando si verifica almeno uno dei seguenti casi:
 - **Anomalie critiche in ricezione** ($rx_status = RX_CRITICAL$): la Board 2 rileva autonomamente un'interruzione del flusso dati.

- **Anomalia critiche del supervisore Board 1** ($b1_sup_status = SUP_CRITICAL$): il monitoraggio remoto della Board 1 fallisce.
- **Richiesta esplicita di switch** ($b1_requests_switch = true$): la Board 1 stessa segnala di non poter proseguire. In base alla funzione `classifyB1CriticalMask`, questa richiesta diventa vera se la Board 1 riscontra un errore critico di comunicazione (RX_BIT) o un timeout del supervisore di Board 2(B2SUP_BIT).

Nonostante la possibilità di comandare il rover, è bene distinguere i casi che consentono o meno l'effettivo comando dei motori da parte di Board 2.

- **Attuazione concreta:** Board 2 invia i segnali di attuazione e comanda effettivamente i motori quando la Board 1 riscontra un errore critico di comunicazione (RX_BIT) o un timeout del supervisore di Board 2(B2SUP_BIT). In questo caso Board 1 apre il rele per permettere al segnale di attuazione di Board 2 di arrivare ai motori. La Figura 68 mostra il caso in cui Board 1 apre il rele a causa di una mancata ricezione da Board 2

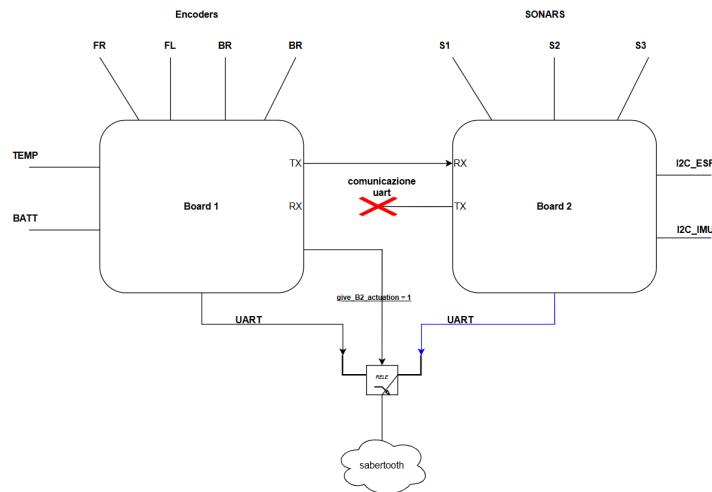


Figure 68: Board 1 non riceve dati dalla Board 2.

- **Attuazione teorica:** Board 2 invia i segnali di attuazione ma non comanda effettivamente i motori quando la Board 2 riscontra un'interruzione nel flusso di dati da Board 1, quindi quando riscontra un errore in ricezione ($rx_status = RX_CRITICAL$). In questo caso Board 2 non può sapere se Board 1 sia ancora in funzione e quindi Board 2 prova comunque ad attuare. Se la comunicazione con Board 1 è ancora stabile,

i messaggi che Board 1 riceve da Board 2 le permettono di continuare ad attuare. La Figura 69 mostra il caso in cui Board 1 non apre il rele nonostante Board 2 non riesca a comunicare con Board 1.

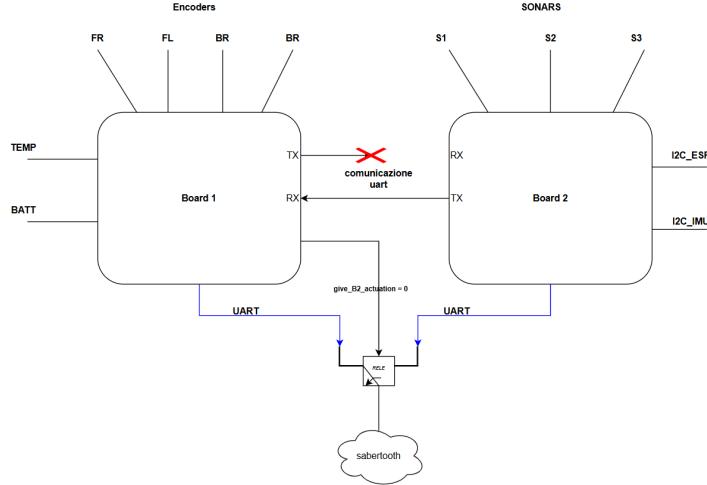


Figure 69: Board 2 non riceve dati dalla Board 1.

9.7 Rilevamento ostacoli

Il comportamento del rover in presenza di ostacoli è regolato sulla base di chi comanda effettivamente i motori, quindi chi è ad attuare:

1. *Board 1 comanda i motori*

- **Distanza dell'ostacolo ≤ 70 cm:** il rover deve fermarsi immediatamente per evitare collisioni.
- **Ostacolo a distanza > 100 cm in movimento tra due sonar:** il rover deve determinare la direzione dell'ostacolo e deve deviare il percorso di conseguenza in direzione del sonar che per prima ha rilevato l'ostacolo.

2. *Board 2 comanda i motori (modalità degradata con Board isolata)*

- **Distanza dell'ostacolo ≤ 300 cm:** il rover deve fermarsi immediatamente per evitare collisioni.

In seguito verranno mostrati i chart realizzati per la gestione delle due casistiche.

9.7.1 Gestione ostacoli con Board 1 attuatrice

Questo tipo di gestione si ha quando il sistema si trova nelle condizioni di funzionamento nominali, quindi quando la Board 1 invia l'attuazione ai motori.

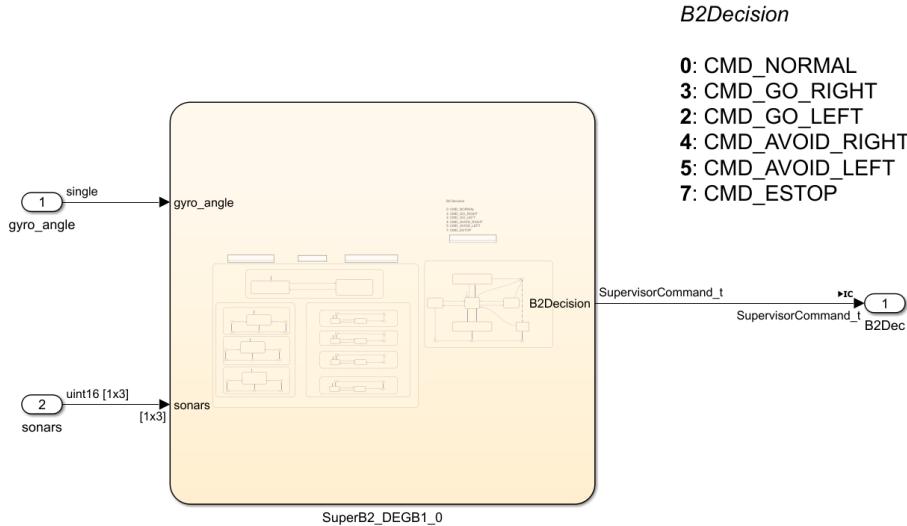


Figure 70: Logica di gestione degli ostacoli in stato non degradato.

Il chart per la gestione degli ostacoli con Board 1 attuatrice è mostrato in Figura 71.

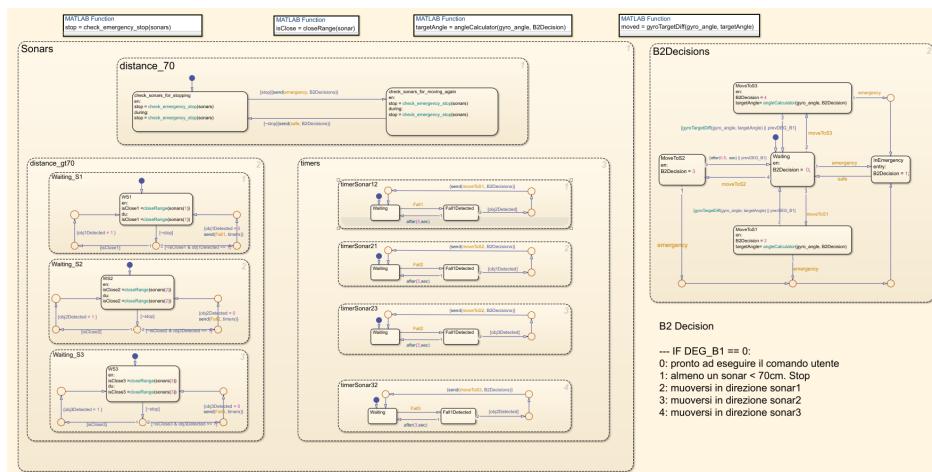
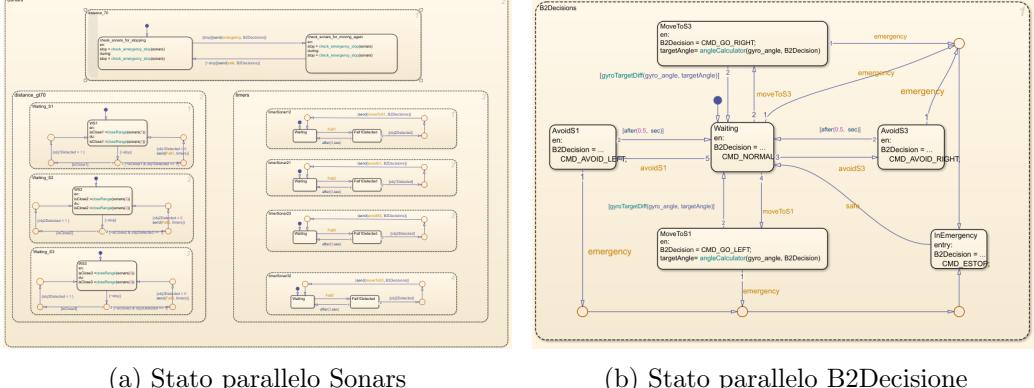


Figure 71: Chart di gestione ostacoli con Board 1 attuatrice.

In particolare, il chart è composto da 2 stati paralleli: *Sonars* e *B2Decisione*

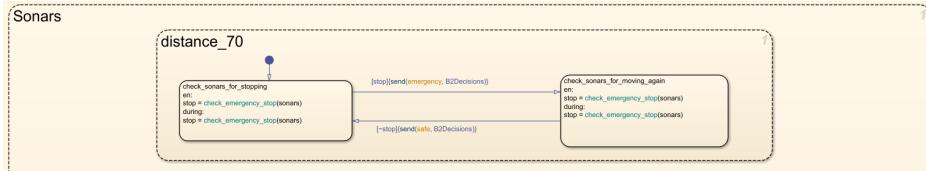


(a) Stato parallelo Sonars

(b) Stato parallelo B2Decisione

Figure 72: Stati paralleli del chart in stato non degradato.

1. **B2Decisions:** Lo stato parallelo *B2Decisione* è dipendente dallo stato *Sonars* in quanto le sue transizioni vengono attivate da segnali provenienti da *Sonars*. In base ai segnali ricevuti, è capace di settare la variabile di output del chart che indica la decisione presa dal supervisore. Quindi nello stato *B2Decisions* si determina l'output del chart, che è un numero che varia da 0 a 4. Gli output determinano le azioni possibili che includono l'arresto immediato del rover o la deviazione del percorso intorno al rover dell'ostacolo. In quest'ultimo caso, la deviazione dura fintanto che il rover non ruota di 45° rispetto alla direzione iniziale, verso quella del sonar che per primo ha rilevato l'ostacolo.
2. **Sonars:** All'interno di questo stato parallelo sono presenti altri 3 stati paralleli.
 - (a) ***distance_70*:** Rappresenta la condizione in cui uno dei sonar rileva un ostacolo a una distanza ≤ 70 cm.

Figure 73: Stato parallelo per la gestione di un ostacolo a distanza ≤ 70 cm.

In questo stato quando uno dei sonar rileva un ostacolo a una distanza inferiore o uguale a 70 cm, viene attivata una transizione

che porta allo stato di arresto immediato del rover. In particolare, quando un sonar rileva la presenza di un ostacolo a distanza ≤ 70 cm, viene inviato un segnale **Emergency** allo stato parallelo *B2Decisione* per fermare il rover. *B2Decision* utilizza questo segnale per portarsi nello stato in cui l'output del chart prevede lo stop.

- (b) ***distance_gt70* —- *timers***: Questi due stati insieme permettono il rilevamento di un ostacolo in movimento tra le coppie di sonar
- *S1-S2* (tra sonar di sinistra e sonar centrale)
 - *S2-S1* (tra sonar centrale e sonar di sinistra)
 - *S2-S3* (tra sonar centrale e sonar di destra)
 - *S3-S2* (tra sonar di destra e sonar centrale)

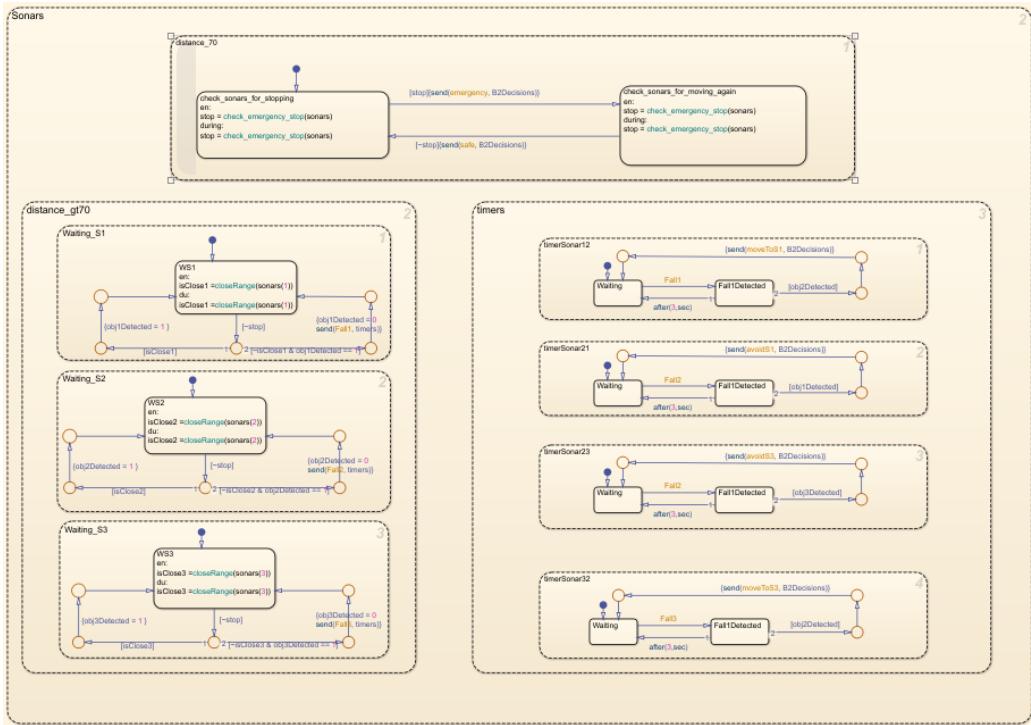


Figure 74: Stato parallelo per la gestione di un ostacolo in movimento a distanza > 70 cm.

Nello stato *distance_gt70* sono presenti 3 stati paralleli, *Waiting_S1*, *Waiting_S2*, *Waiting_S3*, uno per ogni sonar.

Di seguito si analizza la dinamica di rilevamento di un ostacolo che

si sposta dal sonar S_1 verso il sonar S_2 . Tale logica è da considerarsi valida per ogni coppia di sensori precedentemente elencata. Si assume, come condizione necessaria, l'assenza di ostacoli a una distanza inferiore a 70 cm; in caso contrario, il sistema non procederebbe al rilevamento di oggetti in movimento.

- i. **Attivazione (S_1):** Quando il sonar S_1 rileva un oggetto entro il range 100–300 cm, la variabile $obj1Detected$ viene impostata a 1 (**fronte di salita**).
- ii. **Transizione e Timing:** Nel momento in cui l'oggetto esce dal campo d'azione di S_1 , la variabile $obj1Detected$ torna a 0 (**fronte di discesa**). Contestualmente, lo stato `timerSonar12` del modulo `timers` avvia un conteggio di 3 secondi.
- iii. **Verifica (S_2):** Se il sonar S_2 rileva l'ostacolo (sempre tra 100 e 300 cm) entro la finestra temporale dei 3 secondi, viene inviato il segnale `moveToS1` allo stato parallelo `B2Decision`. Qualora il timer scada senza alcun rilevamento da parte di S_2 , non viene trasmesso alcun segnale.

9.7.2 Gestione ostacoli con Board 2 attuatrice

Questo tipo di gestione si verifica quando il controllo passa alla Board 2 (Slave). Ciò avviene quando la Board 1 rileva un errore critico di comunicazione (`RX_BIT`) oppure un timeout del supervisore di Board 2.

Il chart per la gestione degli ostacoli quando è Board 2 ad attuare è mostrato in Figura 75.

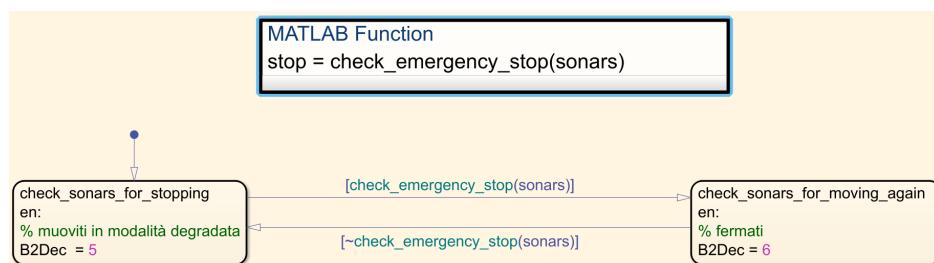


Figure 75: Chart di gestione ostacoli quando B2 attua

In questo caso, la logica di gestione degli ostacoli è semplificata rispetto allo stato non degradato. Infatti, l'unica condizione considerata è la presenza di un ostacolo a una distanza inferiore o uguale a 300 cm. Quando uno dei sonar rileva un ostacolo entro questo range, viene attivata una transizione che porta l'uscita del supervisore all'arresto immediato del rover.

9.8 Attuazione

Il sottosistema *Actuation decision and motion state* viene attivato quando la Board 2 assume la responsabilità dell'attuazione. Questo subentro avviene in caso di interruzione della ricezione dei dati di Board 1 o su richiesta esplicita di quest'ultima (ad esempio, qualora la Board 1 rilevi un guasto critico nella ricezione da Board 2).

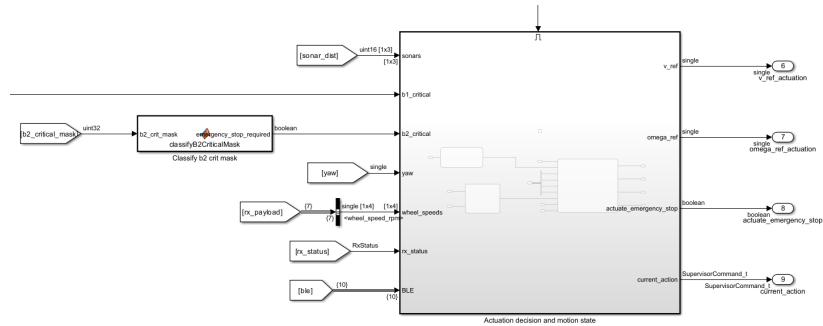


Figure 76: Chart per la gestione dell'attuazione del comando di movimento del rover

In questa modalità, il supervisore gestisce il movimento in regime degradato, limitando la velocità al 50% del valore nominale (pari a 80 RPM rispetto ai 160 RPM massimi). Il sistema impone inoltre un arresto di emergenza immediato al verificarsi di una delle seguenti condizioni:

- rilevazione di un ostacolo a una distanza inferiore ai 3 metri;
- presenza di un guasto critico nella maschera locale di Board 2;
- ricezione di almeno un guasto critico nella critical mask proveniente dalla Board 1 (se il link di comunicazione è ancora attivo).

Oltre alla marcia normale condizionata dalle condizioni descritte sopra, il sottosistema ha anche la possibilità di gestire la rotazione a 180° o la semplice retromarcia, analogamente al sottosistema *Backward_mode_toggle* della Board 1.