

Università degli Studi di Salerno  
Dipartimento di Ingegneria dell'Informazione, Elettrica  
e Matematica Applicata

Project Work Gruppo 2

**C.R.I.C.  
(Cooperative Rover Integrated Controller)**



Nome e Cognome	Matricola	Email
Enrico Cavuoto	0622702565	e.cavuoto1@studenti.unisa.it
Biagio De Martino	0622702559	b.demartino5@studenti.unisa.it
Gianmarco Guerriero	0622702470	g.guerriero23@studenti.unisa.it
Attilio Marco Sessa	0622702468	a.sessa111@studenti.unisa.it

Anno Accademico 2025/2026

# Indice

---

<b>I Introduzione e requisiti</b>	<b>7</b>
<b>1 Introduzione</b>	<b>8</b>
1.1 Obiettivo del progetto . . . . .	8
1.2 Organizzazione hardware . . . . .	8
<b>2 Elicitazione dei requisiti</b>	<b>11</b>
2.1 Requisiti funzionali . . . . .	11
2.1.1 Requisiti di movimento . . . . .	11
2.1.2 Requisiti di controllo . . . . .	11
2.1.3 Requisiti di interfacciamento dei sensori . . . . .	11
2.1.4 Requisiti di attuazione delle luci . . . . .	12
2.1.5 Requisiti di controllo della dinamica di marcia . . . . .	14
2.1.6 Requisiti di limitazione della velocità . . . . .	14
2.1.7 Requisiti di comando . . . . .	15
2.1.8 Requisiti per il rilevamento degli ostacoli . . . . .	16
2.1.9 Requisiti funzionali di sicurezza . . . . .	19
2.1.10 Requisiti per il monitoraggio . . . . .	19
2.2 Requisiti non funzionali . . . . .	21
<b>II Stato e supervisione del sistema</b>	<b>22</b>
<b>3 Analisi degli aspetti critici</b>	<b>23</b>
3.1 Fault Tree Analysis . . . . .	24
3.1.1 FT1 - ROVER CRASH . . . . .	24
3.1.2 FT2 - ROVER UNABLE TO MOVE . . . . .	26
3.2 Strategie di mitigazione e gestione dei fault . . . . .	29
3.2.1 Gestione dei fault delle board di controllo . . . . .	29
3.2.2 Gestione dei fault della catena di attuazione . . . . .	30
3.2.3 Gestione dei fault della sensoristica . . . . .	31
3.2.4 Condizioni energetiche e termiche degradanti . . . . .	33
3.2.5 Sintesi complessiva delle politiche di gestione dei fault . . . . .	34
3.3 Definizione e giustificazione dei tempi critici . . . . .	35
3.3.1 Protezione Termica – Temperatura Massima Ammissibile . . . . .	35
3.3.2 Discriminazione Ostacolo Statico / Ostacolo in Movimento . . . . .	36
<b>4 Progettazione della supervisione</b>	<b>37</b>
4.1 Modalità normale . . . . .	37
4.1.1 Scambio dello Stato Locale . . . . .	38
4.1.2 Scambio dello Stato Globale . . . . .	40
4.1.3 Scambio della Decisione . . . . .	42
4.2 Modalità degradata . . . . .	44
4.3 Modalità single-board . . . . .	46

<b>5 Definizione delle variabili di stato</b>	<b>47</b>
5.1 Enumerazioni . . . . .	47
5.1.1 Controllo e dinamica . . . . .	48
5.1.2 Segnalazione visiva . . . . .	48
5.1.3 Diagnostica e ambiente . . . . .	49
5.2 Bus di Stato . . . . .	51
5.2.1 Stato locale della Board 1 . . . . .	51
5.2.2 Stato locale della Board 2 . . . . .	52
5.2.3 Stato globale . . . . .	53
5.2.4 Decisione . . . . .	53
5.3 Strutture di comunicazione . . . . .	54
<b>6 Modellazione della supervisione</b>	<b>55</b>
6.1 Scelta del protocollo fisico di comunicazione . . . . .	55
6.2 Board 2/ Master . . . . .	56
6.2.1 Variabili del modello . . . . .	57
6.2.2 Struttura architetturale . . . . .	61
6.3 Board 1/ Slave . . . . .	66
6.3.1 Variabili del Modello . . . . .	66
6.3.2 Struttura architetturale . . . . .	70
6.4 Automi comuni . . . . .	74
6.4.1 Moving_obstacle . . . . .	75
6.4.2 Battery_temperature_manager . . . . .	76
6.4.3 Combo . . . . .	78
6.4.4 Board_decision . . . . .	79
<b>7 Generazione del codice</b>	<b>92</b>
7.1 Corrispondenza tra tipi e occupazione in memoria . . . . .	92
7.1.1 Stima dell'occupazione — StateBusB1 . . . . .	93
7.1.2 Stima dell'occupazione — StateBusB2 . . . . .	93
7.1.3 Stima dell'occupazione — GsBus . . . . .	94
7.1.4 Stima dell'occupazione — DecBus . . . . .	94
7.1.5 Considerazioni sull'allineamento . . . . .	94
7.2 Requisiti preliminari alla generazione . . . . .	94
7.2.1 Definizione rigorosa dei tipi di dato . . . . .	95
7.2.2 Configurazione diagnostica . . . . .	95
7.2.3 Definizione delle enumerazioni . . . . .	96
7.3 Configurazione del Solver e criticità temporali . . . . .	96
7.4 Strategia di disaccoppiamento e Stub Functions . . . . .	97
7.5 Script di post-processing . . . . .	98
7.5.1 Esposizione degli stati . . . . .	99
7.5.2 Integrazione automatica nel repository . . . . .	99
7.6 Procedura operativa di generazione . . . . .	100
<b>III Sistema Operativo Real Time</b>	<b>101</b>
<b>8 Schedulazione Real Time</b>	<b>102</b>
8.1 Spazio e tempo di frenata . . . . .	102

8.2	Identificazione del task set . . . . .	104
8.3	Sincronizzazione e dipendenze tra task . . . . .	105
8.4	Dimensionamento dei task . . . . .	107
8.4.1	Analisi generale del supervisionTask . . . . .	107
8.4.2	Board 2 . . . . .	111
8.4.3	Board 1 . . . . .	120
8.5	Scheduling e dimostrazione di schedulabilità . . . . .	126
8.5.1	Scheduling della Board 1 . . . . .	127
8.5.2	Scheduling della Board 2 . . . . .	128
8.6	Stato degradato . . . . .	129
8.7	Dettagli implementativi in FreeRTOS . . . . .	130
8.7.1	Allocazione statica dei task . . . . .	130
8.7.2	Assegnazione delle priorità e schedulazione . . . . .	130
8.7.3	Gestione delle deadline . . . . .	131
<b>IV</b>	<b>Controllo dei motori</b>	<b>132</b>
<b>9</b>	<b>Identificazione dei motori</b>	<b>133</b>
9.1	Modello dinamico del motore in corrente continua . . . . .	133
9.1.1	Modello teorico . . . . .	133
9.1.2	Semplificazione del modello . . . . .	134
9.2	Acquisizione dei dati . . . . .	134
9.3	Elaborazione dei dati in MATLAB . . . . .	135
9.3.1	Caricamento e allineamento dei segnali . . . . .	135
9.3.2	Costruzione dei segnali normalizzati . . . . .	137
9.3.3	Interpolazione temporale e segnali finali . . . . .	138
9.3.4	Visualizzazione dei risultati . . . . .	140
9.4	System Identification Tool di MATLAB . . . . .	141
9.4.1	Apertura dello strumento . . . . .	141
9.4.2	Importazione dei dati nel dominio del tempo . . . . .	141
9.4.3	Verifica visiva e preprocessing . . . . .	142
9.4.4	Selezione della struttura del modello . . . . .	142
9.4.5	Stima del modello . . . . .	143
9.4.6	Validazione del modello . . . . .	143
9.4.7	Espортazione del modello e utilizzo finale . . . . .	143
9.5	Modelli Identificati per i Motori del Rover . . . . .	144
<b>10</b>	<b>Progettazione dei controllori per i motori</b>	<b>145</b>
10.1	Controllo in retroazione e regolazione PID . . . . .	145
10.1.1	Controllo in retroazione . . . . .	145
10.1.2	Struttura del controllore PID . . . . .	145
10.2	Progettazione del Controllore PID tramite <i>Sisotool</i> . . . . .	146
10.2.1	Scelta del controllore: perché un PI e non un PID . . . . .	146
10.2.2	Apertura di Sisotool e configurazione dello strumento . . . . .	146
10.2.3	Criteri adottati per la progettazione dei controllori . . . . .	147
10.2.4	Controllori PI nel continuo . . . . .	149
10.2.5	Scelta del tempo di campionamento . . . . .	149
10.2.6	Implementazione digitale . . . . .	150

10.2.7 Ritardo introdotto . . . . .	151
10.2.8 Temporizzazione dell'algoritmo . . . . .	151
10.2.9 Problematiche di implementazione . . . . .	151
<b>11 Implementazione software</b>	<b>154</b>
11.1 Scelta della modalità di pilotaggio dei driver Sabertooth . . . . .	154
11.2 Calibrazione dei motori per il controllo in anello aperto . . . . .	155
11.2.1 Calcolo della frequenza PWM . . . . .	155
11.2.2 Fase 1: Determinazione del punto di zero . . . . .	156
11.2.3 Fase 2: Determinazione dei fattori di scala . . . . .	156
11.3 Algoritmo di controllo . . . . .	157
11.3.1 Acquisizione del segnale . . . . .	157
11.3.2 Ramping del riferimento . . . . .	158
11.3.3 Calcolo del comando PID . . . . .	160
11.3.4 Applicazione al motore: saturazione duty e calcolo PWM . . . . .	161
11.4 Gestione della sterzata . . . . .	164
<b>12 Comando e gestione remota</b>	<b>166</b>
12.1 Il controller . . . . .	166
12.1.1 Componenti del controller . . . . .	166
12.1.2 Protocollo di comunicazione . . . . .	166
12.1.3 Gestione I/O aggiuntiva . . . . .	167
12.1.4 Alimentazione e monitoraggio . . . . .	167
12.1.5 Interfaccia Grafica Utente (GUI) . . . . .	167
12.2 La ricevente . . . . .	168
12.2.1 Configurazione hardware . . . . .	168
12.2.2 Interfacce di comunicazione . . . . .	168
12.2.3 Architettura firmware . . . . .	169
12.2.4 Elaborazione concorrente . . . . .	170
12.3 Visione e monitoraggio . . . . .	170
12.3.1 Architettura . . . . .	171
12.3.2 Gestione Concorrente con FreeRTOS . . . . .	171
12.3.3 Dashboard per FPV . . . . .	171
<b>V Dettagli implementativi e Test di sistema</b>	<b>172</b>
<b>13 Struttura del Firmware</b>	<b>173</b>
13.1 Driver principali . . . . .	173
13.1.1 Driver della Board1 . . . . .	173
13.1.2 Driver della Board2 . . . . .	174
13.2 Repository . . . . .	175
13.2.1 board1_firmware . . . . .	175
13.2.2 board2_firmware . . . . .	175
<b>14 Test di funzionamento</b>	<b>176</b>
14.1 Test di movimento . . . . .	176
14.1.1 T-RM-4 — Rotazione sul posto . . . . .	178
14.1.2 T-RM-5 — Retromarcia speciale con rotazione di 180° . . . . .	178

14.1.3	T-RM-6 — Frenata progressiva (smooth) . . . . .	179
14.1.4	T-RM-7 — Frenata di emergenza . . . . .	179
14.2	Controllo . . . . .	180
14.2.1	T-RCO-1 — Controllo PID . . . . .	180
14.2.2	T-RCO-2 — Controllo open-loop . . . . .	180
14.3	Attuazione delle luci . . . . .	181
14.3.1	T-RAL-1 — Selezione modalità luci . . . . .	181
14.3.2	T-RAL-2 — Modalità OFF . . . . .	181
14.3.3	T-RAL-3 — Modalità ON . . . . .	182
14.3.4	T-RAL-4 Modalità AUTO . . . . .	182
14.3.5	T-RAL-6 — Traslazione laterale (sinistra / destra) . . . . .	183
14.3.6	T-RAL-7 — Retromarcia speciale . . . . .	184
14.3.7	T-RAL-8 — Segnalazione luminosa modalità . . . . .	184
14.4	Controllo della dinamica di marcia . . . . .	186
14.4.1	T-RCM-1 — Selezione della modalità di dinamica di marcia . . . . .	186
14.4.2	T-RCM-2 — Modalità di default all'accensione . . . . .	186
14.4.3	T-RCM-3 — Influenza della modalità sulla dinamica di variazione della velocità . . . . .	187
14.5	Limitazione della velocità . . . . .	187
14.5.1	T-RLM-1 — Limiti di velocità di default . . . . .	187
14.5.2	T-RLM-2 — Regolazione dei limiti di velocità . . . . .	188
14.6	Rilevamento degli ostacoli . . . . .	188
14.6.1	T-RRO-1 — Configurazione e portata dei sensori . . . . .	188
14.6.2	T-RRO-2 — Modalità di sensibilità . . . . .	189
14.6.3	T-RRO-3 — Ostacolo frontale progressivo (prudente) . . . . .	189
14.6.4	T-RRO-4 — Ostacolo improvviso ravvicinato (prudente) . . . . .	189
14.6.5	T-RRO-5 — Ostacolo laterale anticipato (prudente) . . . . .	190
14.6.6	T-RRO-6 — Ostacolo improvviso (rilassata) . . . . .	190
14.7	Sicurezza . . . . .	191
14.7.1	T-RFS-1 — Ingresso in modalità degradata . . . . .	191
14.7.2	T-RFS-2 — Limitazione della velocità in modalità degradata . . . . .	191
14.7.3	T-RFS-3 — Rilevamento ostacoli in modalità degradata . . . . .	192
14.7.4	T-RFS-4 — Ingresso in modalità single-board . . . . .	192
<b>A</b>	<b>Printed Circuit Board</b>	<b>195</b>
A.1	PCB di controllo sicuro dei driver di potenza . . . . .	195
A.1.1	Problema di sicurezza nei driver Sabertooth . . . . .	195
A.1.2	Strategia di avvio sicuro dei driver . . . . .	195
A.1.3	Selezione della sorgente di controllo . . . . .	195
A.1.4	Progettazione e realizzazione del circuito stampato . . . . .	196
A.1.5	Interfaccia di collegamento e distribuzione dei segnali . . . . .	196
A.2	PCB HAT per le board NUCLEO . . . . .	199
A.2.1	PCB HAT della scheda Master . . . . .	199
A.2.2	PCB HAT della scheda Slave . . . . .	199

<b>B Componenti meccanici stampati in 3D</b>	<b>201</b>
B.1 Controller del rover . . . . .	201
B.2 Case per le board di controllo STM32G4 . . . . .	202
B.3 Supporto per PCB di controllo dei motori . . . . .	202
B.4 Supporto per i sensori sonar . . . . .	203
B.5 Supporto per ESP32-CAM . . . . .	204
B.5.1 Supporto per antenna esterna . . . . .	205
<b>C Configurazione delle schede e schemi di collegamento</b>	<b>206</b>
C.1 Board 1 (Slave) . . . . .	206
C.1.1 Configurazione . . . . .	206
C.1.2 Tabella di assegnazione dei pin . . . . .	207
C.2 Board 2 (Master) . . . . .	208
C.2.1 Configurazione . . . . .	208
C.2.2 Tabella di assegnazione dei pin . . . . .	209
C.2.3 Schema di collegamento complessivo . . . . .	210

# **Parte I**

## **Introduzione e requisiti**

# Introduzione

## 1.1 Obiettivo del progetto

Il progetto si pone l'obiettivo di realizzare un rover a guida remota radiocontrollato. Il veicolo risulta dotato di 4 ruote motrici e nessun asse di sterzo. Il rover deve essere in grado di muoversi nello spazio, controllando in maniera indipendente i 4 motori ed evitando opportunamente gli ostacoli attraverso la sensoristica di cui è dotato. In particolare, il veicolo monta 3 sensori ad ultrasuoni HC-SR04 posizionati frontalmente e lateralmente con un'angolazione di 45 gradi. Inoltre, durante il funzionamento, l'elettronica di bordo deve essere in grado di monitorare parametri di funzionamento quali temperatura di esercizio e tensione dell'accumulatore. La progettazione ha come principale scopo la realizzazione di un sistema resiliente a condizioni anomale di funzionamento, adottando principi di ridondanza e diversità per ridurre il rischio di fallimento e gestire modalità di funzionamento degradato.

## 1.2 Organizzazione hardware

Come mostrato in Figura 1.1, l'architettura di sistema prevede l'adozione di due schede di controllo, alle quali sono connesse le diverse periferiche di cui il rover è dotato.

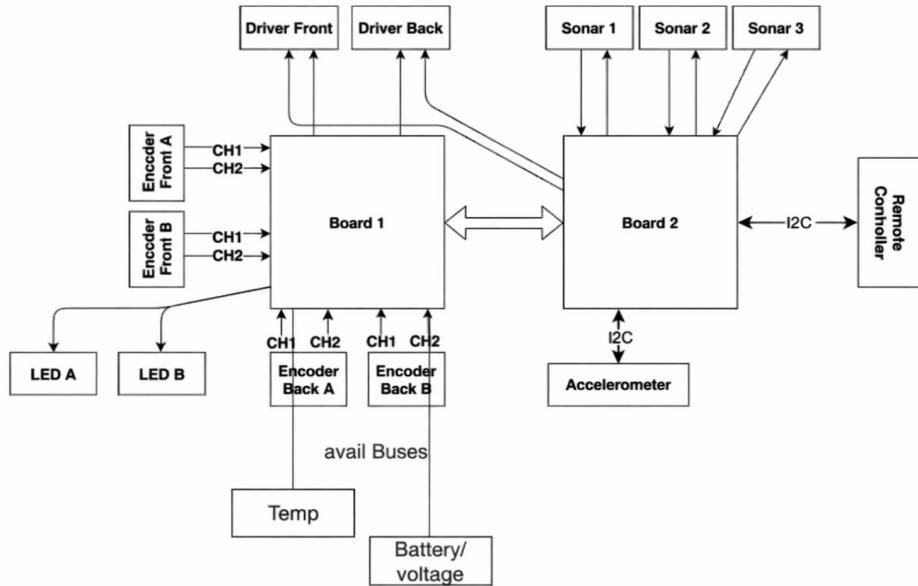


Figura 1.1: Schema hardware

### Board 1

La *Board 1* è dotata di un insieme eterogeneo di sensori e attuatori che la rendono il nodo centrale per il controllo della locomozione e il monitoraggio dello stato del veicolo. In particolare, essa integra:

- **Encoder dei motori:** ciascuno dei quattro motori brushed indipendenti *36GP-540-51* è equipaggiato con un encoder magnetico a due canali, con risoluzione pari a 12 PPR e frequenza massima di 800 kHz, utilizzato per il rilevamento della velocità e della posizione;
- **Motor driver:** due driver *Sabertooth 2X12*, impiegati per l'attuazione dei segnali di comando ai quattro motori;
- **Moduli LED anteriori:** due unità *A4WD3* con luci bianche ad alta potenza e luci rosse;
- **Monitoraggio della batteria:** ingresso dedicato alla misura della tensione di alimentazione (valore al momento trattato in forma generica);
- **Sensore di temperatura interna:** misura della temperatura tramite il sensore integrato nel microcontrollore *STM32G474RE*.

Grazie a questa dotazione, la Board 1 è l'unica in grado di eseguire un controllo in retroazione completo sulla trazione del veicolo, poiché può acquisire direttamente il feedback dagli encoder e modulare in tempo reale i segnali inviati ai motor driver. Oltre alla regolazione della velocità e della posizione dei motori, essa svolge funzioni di supervisione dello stato energetico e termico, permettendo di mantenere condizioni operative sicure e stabili anche durante manovre complesse o prolungate.

## Board 2

La Board 2 è progettata per la percezione dell'ambiente circostante e per la gestione dell'interazione con l'utente. È dotata di un insieme di sensori dedicati al rilevamento degli ostacoli e alla stima dello stato dinamico del veicolo, oltre a fornire un canale di comunicazione per i comandi remoti. In particolare, la scheda integra:

- **Sensori di prossimità:** tre moduli a ultrasuoni *HC-SR04*, posizionati rispettivamente sul lato sinistro, sul lato destro e frontalmente, utilizzati per il rilevamento degli ostacoli e il supporto alle strategie di navigazione sicura;
- **Accelerometro:** un'unità *MPU6050*, con interfaccia di comunicazione  $I^2C$ , impiegata per acquisire accelerazioni e valori inerziali utili all'analisi dell'assetto dinamico del veicolo;
- **Interfaccia di comando remoto:** collegamento con un controller esterno tramite tecnologia  $I^2C$  utilizzato per ricevere input e modalità operative impostate dall'utente;
- **Collegamento diretto ai motor driver:** disponibile per la gestione dei motori in modalità di funzionamento degradato, quando la Board 1 non è disponibile; tale attuazione avviene senza feedback encoder.

Grazie a questa configurazione, la Board 2 svolge un ruolo centrale nella percezione dell'ambiente e nella raccolta dei comandi dell'utente. Essa fornisce informazioni essenziali per il rilevamento degli ostacoli e per la valutazione dello stato dinamico, contribuendo ai processi decisionali del sistema. Inoltre, in caso di criticità o malfunzionamenti della Board 1, garantisce la continuità operativa assumendo il controllo di base dei motori attraverso la modalità degradata.

### Interconnessione tra le due schede

Le due schede sono connesse mediante un collegamento bidirezionale che consente lo scambio continuo delle rispettive informazioni locali. Tale integrazione permette a ciascuna scheda di costruire una rappresentazione coerente dello *stato globale* del veicolo, condizione necessaria per implementare:

- algoritmi di decisione distribuita;
- ridondanza logica e operativa;
- una più elevata affidabilità complessiva del sistema.

L'interconnessione garantisce quindi che il veicolo possa continuare a operare anche in presenza di guasti parziali, grazie al supporto reciproco tra i due sottosistemi.

---

# Elicitazione dei requisiti

---

## 2.1 Requisiti funzionali

La presente sezione descrive in modo sistematico i requisiti funzionali del rover, organizzati per ambiti di competenza del sistema.

### 2.1.1 Requisiti di movimento

Si definiscono di seguito i requisiti relativi al movimento del rover in risposta ai comandi provenienti dal controller remoto. In particolare, vengono specificati la gestione della velocità, il comportamento dei motori durante gli spostamenti lineari e rotazionali, nonché la loro modalità operativa all'avvio.

- **RM.1:** Il rover deve poter traslare in avanti in linea retta.
- **RM.2:** Il rover deve poter eseguire una retromarcia normale traslando indietro in linea retta.
- **RM.3:** Il rover deve poter ruotare, consentendo una traiettoria curva, mentre trasla avanti o indietro.
- **RM.4:** Il rover da fermo deve poter ruotare intorno al suo baricentro.
- **RM.5:** Il rover deve poter eseguire una retromarcia speciale ruotando di 180 gradi.
- **RM.6:** Il rover deve supportare una frenata smooth, riducendo in maniera progressiva la velocità fino all'arresto.
- **RM.7:** Il rover deve supportare una frenata di emergenza, arrestando istantaneamente la marcia

### 2.1.2 Requisiti di controllo

- **RCO.1:** In condizioni operative normali, i motori devono essere controllati in retroazione attraverso un PID dedicato.
  - **RCO.1.1:** Il controllo dei motori tramite PID deve essere delegato alla Board 1.
- **RCO.2:** In caso di malfunzionamenti del controllo di retroazione, i motori devono essere controllati in open-loop.
  - **RCO.2.1:** Il controllo dei motori in open-loop deve essere delegato alla Board 2.

### 2.1.3 Requisiti di interfacciamento dei sensori

- **RIS.1:** La Board 1 deve leggere la temperatura fornita dal suo sensore di temperatura interno.
- **RIS.2:** La Board 1 deve leggere il valore di voltaggio della batteria ad essa collegata

- **RIS.3:** La Board 1 deve leggere la velocità di rotazione del motore anteriore A mediante il corrispettivo encoder.
- **RIS.4:** La Board 1 deve leggere la velocità di rotazione del motore anteriore B mediante il corrispettivo encoder.
- **RIS.5:** La Board 1 deve leggere la velocità di rotazione del motore posteriore A mediante il corrispettivo encoder.
- **RIS.6:** La Board 1 deve leggere la velocità di rotazione del motore posteriore B mediante il corrispettivo encoder.
- **RIS.7:** La Board 2 deve acquisire la distanza da eventuali ostacoli mediante il sensore a ultrasuoni 1.
- **RIS.8:** La Board 2 deve acquisire la distanza da eventuali ostacoli mediante il sensore a ultrasuoni 2.
- **RIS.9:** La Board 2 deve acquisire la distanza da eventuali ostacoli mediante il sensore a ultrasuoni 3.
- **RIS.10:** La Board 2 deve leggere le misure di accelerazione e velocità angolare provenienti dallo giroscopio-accelerometro.
- **RIS.11:** La Board 2 deve ricevere i comandi inviati dal controller remoto wireless.

#### 2.1.4 Requisiti di attuazione delle luci

- **RAL.1:** In condizioni operative normali le luci devono poter essere operative in tre modalità selezionabili: **OFF**, **ON**, **AUTO**.
- **RAL.2:** In condizioni operative normali la pressione del pulsante associato alla levetta analogica destra deve far avanzare ciclicamente le modalità operative delle luci seguendo la sequenza **OFF->ON->AUTO**.
- **RAL.3:** In condizioni operative normali quando è selezionata la modalità **OFF**, tutte le luci devono essere spente.
- **RAL.4:** In condizioni operative normali quando il rover è fermo, le luci frontali devono avere comportamenti dipendenti dalla modalità selezionata.
  - **RAL.4.1:** Quando è selezionata la modalità **ON**, le luci frontali devono rimanere accese bianche.
  - **RAL.4.2:** Quando è selezionata la modalità **AUTO**, le luci frontali devono rimanere spente.
- **RAL.5:** In condizioni operative normali quando il rover è fermo e la modalità **OFF** non è selezionata, la striscia di LED posteriore deve visualizzare una luce di posizione rossa, accendendo quattro LED sul lato sinistro e quattro LED sul lato destro.
- **RAL.6:** In condizioni operative normali quando il rover è in movimento in avanti e con modalità diversa da **OFF**, le luci devono notificare l'azione.

- **RAL.6.1:** Le luci frontali devono essere accese bianche.
- **RAL.6.2:** La striscia di LED posteriore deve visualizzare una luce di posizione rossa, accendendo quattro LED sul lato sinistro e quattro LED sul lato destro.
- **RAL.7:** In condizioni operative normali quando il rover sta frenando e *non* è selezionata la modalità **OFF**, le luci devono notificare l’azione.
  - **RAL.7.1:** Le luci frontali devono essere accese bianche.
  - **RAL.7.2:** Le luci posteriori devono essere accese rosse.
- **RAL.8:** In condizioni operative normali quando il rover sta traslando verso sinistra e *non* è selezionata la modalità **OFF**, le luci devono notificare l’azione.
  - **RAL.8.1:** La luce frontale sinistra deve lampeggiare di colore rosso.
  - **RAL.8.2:** La luce frontale destra deve essere accesa di colore bianco.
  - **RAL.8.3:** Le luci posteriori devono simulare un’animazione che riempie progressivamente metà dei led verso sinistra.
- **RAL.9:** In condizioni operative normali quando il rover sta traslando verso destra e *non* è selezionata la modalità **OFF**, le luci devono notificare l’azione.
  - **RAL.9.1:** La luce frontale destra deve lampeggiare di colore rosso.
  - **RAL.9.2:** La luce frontale sinistra deve essere accesa di colore bianco.
  - **RAL.9.3:** Le luci posteriori devono simulare un’animazione che riempie progressivamente metà dei led verso destra.
- **RAL.10:** In condizioni operative normali quando il rover si sta muovendo all’indietro e *non* è selezionata la modalità **OFF**, le luci devono notificare l’azione.
  - **RAL.10.1:** Le luci frontali devono essere accese bianche.
  - **RAL.10.2:** L’intera striscia di led posteriori deve essere accessa bianca.
- **RAL.11:** Quando il sistema non è nello stato normale di funzionamento, le luci devono notificare tale situazione.
- **RAL.12:** In condizioni operative normali durante l’esecuzione della retromarcia speciale, il sistema di illuminazione del rover deve fornire una segnalazione visiva coerente con la modalità luci selezionata.
  - **RAL.12.1:** Quando la modalità luci è impostata su **AUTO** o **ON**, la striscia LED posteriore deve visualizzare un’animazione dinamica multicolore a effetto arcobaleno, mentre le luci anteriori devono lampeggiare di colore bianco.
  - **RAL.12.2:** Quando la modalità luci è impostata su **OFF**, tutte le luci del rover devono rimanere spente.
- **RAL.13:** In condizioni operative normali quando è selezionata una modalità di controllo della dinamica di marcia, la targhetta posteriore deve assumere un colore associato alla modalità attiva.

- **RAL.13.1:** Il colore associato alla modalità **DEFAULT** deve essere distinto da quello associato alle altre modalità.
- **RAL.13.2:** Il colore associato alla modalità **ECO** deve essere distinto da quello associato alle altre modalità.
- **RAL.13.3:** Il colore associato alla modalità **SPORT** deve essere distinto da quello associato alle altre modalità.
- **RAL.14:** Quando il rover si trova in una condizione operativa anomala, la targhetta posteriore deve notificare tale condizione tramite un colore RGB distinto da quelli utilizzati per le modalità di dinamica di marcia.

### 2.1.5 Requisiti di controllo della dinamica di marcia

- **RCM.1:** In condizioni operative normali il rover deve supportare tre modalità di controllo della dinamica di marcia, denominate **DEFAULT**, **ECO** e **SPORT**.
- **RCM.2:** In condizioni operative normali la pressione del pulsante associato alla levetta analogica sinistra deve consentire il cambio ciclico delle modalità di controllo secondo la sequenza **DEFAULT** → **SPORT** → **ECO**.
- **RCM.3:** In condizioni operative normali la modalità di controllo selezionata deve influenzare la dinamica di variazione della velocità del rover, determinandone la rapidità di risposta.
- **RCM.4:** In condizioni operative normali quando è selezionata la modalità **DEFAULT**, la variazione della velocità del rover deve seguire una dinamica standard.
- **RCM.5:** In condizioni operative normali quando è selezionata la modalità **ECO**, la variazione della velocità del rover deve avvenire in modo più graduale e conservativo.
- **RCM.6:** In condizioni operative normali quando è selezionata la modalità **SPORT**, la variazione della velocità del rover deve avvenire in modo più rapido e reattivo.
- **RCM.7:** All'accensione del rover, in condizioni operative normali, la modalità di controllo attiva deve essere impostata di default su **DEFAULT**.

### 2.1.6 Requisiti di limitazione della velocità

- **RLM.1:** In condizioni operative normali il rover deve supportare una funzionalità di regolazione della velocità di marcia entro limiti minimo e massimo scelti opportunamente.
- **RLM.2:** In condizioni operative normali la regolazione della velocità deve avvenire tramite una combinazione di comandi provenienti dal controller remoto.
- **RLM.3:** In condizioni operative normali tramite la combinazione di comando dedicata, l'utente deve poter incrementare progressivamente la velocità massima consentita fino al valore massimo ammesso dal sistema.

- **RLM.3.1:** La velocità massima impostabile non deve superare la velocità massima nominale prevista per il rover.
- **RLM.4:** In condizioni operative normali tramite la combinazione di comando dedicata, l'utente deve poter decrementare progressivamente la velocità massima consentita fino al valore minimo ammesso dal sistema.
- **RLM.4.1:** La velocità minima impostabile deve essere non negativa.
- **RLM.5:** In condizioni operative normali quando la velocità massima impostata viene ridotta al di sotto della velocità di marcia corrente, il rover deve ridurre la velocità in modo controllato fino a rientrare nel nuovo limite impostato.
- **RLM.6:** All'accensione del rover, in condizioni operative normali, i limiti di velocità devono essere impostati a valori di default.

### 2.1.7 Requisiti di comando

Si definiscono di seguito i requisiti relativi all'acquisizione dei comandi da parte del rover tramite controller remoto. In particolare, vengono specificati la modalità di input, il comportamento della levetta e l'utilizzo dei pulsanti.

- **RC.1:** I comandi di movimento devono essere inviati in modalità analogica tramite l'ausilio di due levette, una dedicata al controllo della velocità e una dedicata al controllo della sterzata.
  - **RC.1.1:** L'angolo di inclinazione nullo per la levetta dedicata al controllo della velocità deve essere interpretato come una velocità nulla.
  - **RC.1.2:** L'angolo di inclinazione nullo per la levetta dedicata al controllo della sterzata deve essere interpretato come una sterzata nulla.
  - **RC.1.3:** L'ampiezza della flessione verticale della levetta dedicata al controllo della velocità rispetto alla posizione centrale deve essere interpretata come la velocità con cui il rover deve muoversi.
  - **RC.1.4:** L'ampiezza della flessione orizzontale della levetta dedicata al controllo della sterzata rispetto alla posizione centrale deve essere interpretata come l'angolo di sterzata con cui il rover deve girare.
  - **RC.1.5:** L'ampiezza della flessione orizzontale della levetta dedicata al controllo della sterzata rispetto alla posizione centrale nel caso il rover sia fermo deve essere interpretata come un comando di rotazione intorno al baricentro.
  - **RC.1.6:** In condizioni operative normali, quando il rover ha la modalità di retromarcia speciale è attivata, con la levetta dedicata alla sterzata in posizione neutra e la levetta dedicata al controllo della velocità inclinata verso il basso, il rover deve eseguire una rotazione di 180°.
- **RC.2:** All'accensione del rover, in condizioni operative normali, di default la modalità di retromarcia speciale deve essere attivata.

- **RC.3:** Il rover deve prevedere un comando di frenata di emergenza, attivabile tramite un pulsante del controller remoto.
- **RC.4:** Il rover deve prevedere un comando di frenata smooth, attivabile tramite un pulsante del controller remoto.
- **RC.5:** In condizioni operative normali, il rover deve prevedere un comando di cambio della modalità di retromarcia, attivabile tramite una combo di pulsanti del controller remoto.

### 2.1.8 Requisiti per il rilevamento degli ostacoli

Si definiscono di seguito i requisiti relativi alla percezione dell'ambiente circostante da parte del rover mediante i sensori a ultrasuoni. In particolare, vengono specificate le condizioni di rilevamento, il comportamento del rover in presenza di ostacoli e le manovre da attuare a seconda della combinazione di attivazioni dei sensori a ultrasuoni.

- **RRO.1:** Il sensore a ultrasuoni 1 deve essere orientato a 45 gradi a sinistra rispetto alla parte anteriore del rover e rilevare ostacoli ad almeno 3 metri di distanza.
- **RRO.2:** Il sensore a ultrasuoni 2 deve essere orientato frontalmente rispetto alla parte anteriore del rover e rilevare ostacoli ad almeno 3 metri di distanza.
- **RRO.3:** Il sensore a ultrasuoni 3 deve essere orientato a 45 gradi a destra rispetto alla parte anteriore del rover e rilevare ostacoli ad almeno 3 metri di distanza.
- **RRO.4:** In condizioni operative normali, il rover deve supportare due modalità di sensibilità per il rilevamento degli ostacoli mediante sensori a ultrasuoni, differenziate in base al livello di prudenza.
  - **RRO.4.1:** Il passaggio tra le modalità di sensibilità deve essere attivabile tramite una combinazione di comandi del controller remoto.
- **RRO.5:** In condizioni operative normali quando il rover è in modalità prudente, il rover deve fermare i motori con una dinamica smooth nel caso in cui venga identificato un ostacolo fermo nel senso di marcia anteriore a una distanza compresa tra 70 cm e 3 m.
- **RRO.6:** In condizioni operative normali quando il rover è in modalità prudente e viene identificato un ostacolo improvviso a una distanza inferiore a 70 cm sul sensore a ultrasuoni 1, il rover deve frenare di emergenza.
  - **RRO.6.1:** Quando il rover è in modalità prudente e il sensore a ultrasuoni 3 non identifica un ostacolo a una distanza inferiore a 70 cm, il rover deve ruotare di 45° verso destra.
  - **RRO.6.2:** Quando il rover è in modalità prudente e il sensore a ultrasuoni 3 identifica un ostacolo a una distanza inferiore a 70 cm e il sensore a ultrasuoni 2 non identifica un ostacolo a tale distanza, il rover deve restare fermo.
  - **RRO.6.3:** Quando il rover è in modalità prudente e entrambi i sensori a ultrasuoni 2 e 3 identificano un ostacolo a una distanza inferiore a 70 cm, il rover deve restare fermo.

- **RRO.7:** In condizioni operative normali quando il rover è in modalità prudente e viene identificato un ostacolo improvviso a una distanza inferiore a 70 cm sul sensore a ultrasuoni 2, il rover deve frenare di emergenza.
- **RRO.7.1:** Quando il rover è in modalità prudente e il sensore a ultrasuoni 3 non identifica un ostacolo a una distanza inferiore a 70 cm, il rover deve ruotare di 45° verso destra.
- **RRO.7.2:** Quando il rover è in modalità prudente e il sensore a ultrasuoni 3 identifica un ostacolo a una distanza inferiore a 70 cm e il sensore a ultrasuoni 1 non identifica un ostacolo a tale distanza, il rover deve ruotare di 45° verso sinistra.
- **RRO.7.3:** Quando il rover è in modalità prudente e entrambi i sensori a ultrasuoni 1 e 3 identificano un ostacolo a una distanza inferiore a 70 cm, il rover deve restare fermo.
- **RRO.8:** In condizioni operative normali quando il rover è in modalità prudente e viene identificato un ostacolo improvviso a una distanza inferiore a 70 cm sul sensore a ultrasuoni 3, il rover deve frenare di emergenza.
- **RRO.8.1:** Quando il rover è in modalità prudente e il sensore a ultrasuoni 1 non identifica un ostacolo a una distanza inferiore a 70 cm, il rover deve ruotare di 45° verso sinistra.
- **RRO.8.2:** Quando il rover è in modalità prudente e il sensore a ultrasuoni 1 identifica un ostacolo a una distanza inferiore a 70 cm e il sensore a ultrasuoni 2 non identifica un ostacolo a tale distanza, il rover deve restare fermo.
- **RRO.8.3:** Quando il rover è in modalità prudente e entrambi i sensori a ultrasuoni 1 e 2 identificano un ostacolo a una distanza inferiore a 70 cm, il rover deve restare fermo.
- **RRO.9:** In condizioni operative normali in modalità prudente, il rover non deve avanzare quando, da fermo, identifica un ostacolo frontalmente a una distanza inferiore a 3 m.
- **RRO.10:** In condizioni operative normali quando il rover è in modalità prudente e il sensore a ultrasuoni 1 rileva per primo un ostacolo a una distanza compresa tra 1.5 m e 3 m, ed entro 1000 ms l'ostacolo è rilevato dal sensore a ultrasuoni 2 nello stesso intervallo di distanza, il rover deve ruotare di 45° verso sinistra senza interrompere la marcia.
- **RRO.11:** In condizioni operative normali quando il rover è in modalità prudente e il sensore a ultrasuoni 3 rileva per primo un ostacolo a una distanza compresa tra 1.5 m e 3 m, ed entro 1000 ms l'ostacolo è rilevato dal sensore a ultrasuoni 2 nello stesso intervallo di distanza, il rover deve ruotare di 45° verso destra senza interrompere la marcia.
- **RRO.12:** In condizioni operative normali quando il rover è in modalità prudente e entrambi i sensori a ultrasuoni 1 e 3 rilevano per primi un ostacolo a una distanza compresa tra 1.5 m e 3 m, ed entro 1000 ms l'ostacolo è rilevato dal sensore a

ultrasuoni 2 nello stesso intervallo di distanza, il rover deve frenare in maniera smooth.

- **RRO.13:** In condizioni operative normali quando il rover è in modalità prudente e il sensore a ultrasuoni 2 rileva un ostacolo a una distanza inferiore a 3 m, il rover deve frenare in maniera smooth.
- **RRO.14:** In condizioni operative normali quando il rover è in modalità rilassata e viene identificato un ostacolo improvviso a una distanza inferiore a 40 cm sul sensore a ultrasuoni 1, il rover deve frenare di emergenza.
  - **RRO.14.1:** Quando il rover è in modalità rilassata e il sensore a ultrasuoni 3 non identifica un ostacolo a una distanza inferiore a 40 cm, il rover deve ruotare di 45° verso destra.
  - **RRO.14.2:** Quando il rover è in modalità rilassata e il sensore a ultrasuoni 3 identifica un ostacolo a una distanza inferiore a 40 cm e il sensore a ultrasuoni 2 non identifica un ostacolo a tale distanza, il rover deve restare fermo.
  - **RRO.14.3:** Quando il rover è in modalità rilassata e entrambi i sensori a ultrasuoni 2 e 3 identificano un ostacolo a una distanza inferiore a 40 cm, il rover deve restare fermo.
- **RRO.15:** In condizioni operative normali quando il rover è in modalità rilassata e viene identificato un ostacolo improvviso a una distanza inferiore a 40 cm sul sensore a ultrasuoni 2, il rover deve frenare di emergenza.
  - **RRO.15.1:** Quando il rover è in modalità rilassata e il sensore a ultrasuoni 3 non identifica un ostacolo a una distanza inferiore a 40 cm, il rover deve ruotare di 45° verso destra.
  - **RRO.15.2:** Quando il rover è in modalità rilassata e il sensore a ultrasuoni 3 identifica un ostacolo a una distanza inferiore a 40 cm e il sensore a ultrasuoni 1 non identifica un ostacolo a tale distanza, il rover deve ruotare di 45° verso sinistra.
  - **RRO.15.3:** Quando il rover è in modalità rilassata e entrambi i sensori a ultrasuoni 1 e 3 identificano un ostacolo a una distanza inferiore a 40 cm, il rover deve restare fermo.
- **RRO.16:** In condizioni operative normali quando il rover è in modalità rilassata e viene identificato un ostacolo improvviso a una distanza inferiore a 40 cm sul sensore a ultrasuoni 3, il rover deve frenare di emergenza.
  - **RRO.16.1:** Quando il rover è in modalità rilassata e il sensore a ultrasuoni 1 non identifica un ostacolo a una distanza inferiore a 40 cm, il rover deve ruotare di 45° verso sinistra.
  - **RRO.16.2:** Quando il rover è in modalità rilassata e il sensore a ultrasuoni 1 identifica un ostacolo a una distanza inferiore a 40 cm e il sensore a ultrasuoni 2 non identifica un ostacolo a tale distanza, il rover deve restare fermo.

- **RRO.16.3:** Quando il rover è in modalità rilassata e entrambi i sensori a ultrasuoni 1 e 2 identificano un ostacolo a una distanza inferiore a 40 cm, il rover deve restare fermo.
- **RRO.17:** In caso di malfunzionamento della board 2, il rover deve fermare i motori ogni volta che il sensore centrale rileva un ostacolo a meno di 3 m oppure i sensori laterali rilevano un ostacolo a meno di 1.5m.

## 2.1.9 Requisiti funzionali di sicurezza

Si definiscono di seguito i requisiti funzionali relativi al comportamento del rover in condizioni di errore, degrado o malfunzionamento dei componenti. In particolare vengono specificate le azioni di sicurezza che il sistema rover deve intraprendere per garantire, ove possibile, il mantenimento di una guida in modalità degradata, o altrimenti l'arresto in condizioni critiche.

- **RFS.1:** In caso di malfunzionamento della Board 2, la Board 1 deve assumere il controllo esclusivo dei motori e fermarli.
- **RFS.2:** In caso di malfunzionamento della Board 1, la Board 2 deve assumere il controllo esclusivo dei motori
  - **RFS.2.1:** La Board 2 deve limitare la velocità a un massimo di 80 rpm.
- **RFS.3:** Quando la temperatura interna della Board 1 raggiunge 100 °C per almeno 15 s, la velocità dei motori deve essere limitata a un valore massimo pari a 80 rpm.
- **RFS.4:** Quando la percentuale di carica della batteria connessa alla Board 1 scende al di sotto di 23%, la velocità dei motori deve essere limitata a un valore massimo pari a 80 rpm.
- **RFS.5:** In caso di malfunzionamento di entrambe le Board, il rover deve arrestarsi.
- **RFS.6:** Quando almeno uno dei due motor controller non risponde ai comandi, il rover deve arrestarsi.
- **RFS.7:** Quando almeno uno degli encoder non funziona, il rover deve arrestarsi.
- **RFS.8:** In condizioni operative normali quando l'accelerometro/giroscopio non risponde alle richieste I2C, il rover deve comunque completare le routine di rotazione adottando una modalità di esecuzione alternativa.
- **RFS.9:** Quando il sistema perde connessione con il controller, il rover deve arrestarsi.
- **RFS.10:** Quando la batteria del controller è al di sotto del 5% il rover deve arrestarsi.
- **RFS.11:** Quando i motori non rispondono ai comandi, il rover deve arrestarsi.

## 2.1.10 Requisiti per il monitoraggio

- **RDM.1:** In condizioni operative normali il sistema deve fornire una dashboard per la visualizzazione remota delle informazioni telemetriche del rover.

- **RDM.1.1:** La dashboard deve visualizzare lo stato di carica della batteria del rover.
  - **RDM.1.2:** La dashboard deve visualizzare la velocità di rotazione dei motori del rover.
  - **RDM.1.3:** La dashboard deve visualizzare il valore della velocità massima di marcia attualmente impostata.
  - **RDM.1.4:** La dashboard deve visualizzare la temperatura di funzionamento del rover.
  - **RDM.1.5:** La dashboard deve visualizzare le distanze rilevate dai sensori a ultrasuoni anteriori.
  - **RDM.1.6:** La dashboard deve visualizzare la modalità di illuminazione abilitata.
  - **RDM.1.7:** La dashboard deve visualizzare la modalità di percezione degli ostacoli attualmente abilitata.
  - **RDM.1.8:** La dashboard deve visualizzare la modalità di dinamica di marcia attualmente attiva.
  - **RDM.1.9:** La dashboard deve visualizzare lo stato della supervisione del sistema.
  - **RDM.1.10:** La dashboard deve consentire la visualizzazione in tempo reale del flusso video proveniente dal sistema di visione a bordo del rover.
  - **RDM.1.11:** La dashboard deve poter operare in assenza di infrastrutture di rete esterne, consentendo il collegamento diretto al sistema rover.
  - **RDM.1.12:** La dashboard deve poter operare in assenza di infrastrutture di rete esterne, consentendo il collegamento diretto al sistema rover.
- **RDM.2:** In condizioni operative normali il controller remoto deve ricevere periodicamente le informazioni telemetriche trasmesse dal rover.
  - **RDM.2.1:** Il controller remoto deve visualizzare lo stato di carica della batteria del rover.
  - **RDM.2.2:** Il controller remoto deve visualizzare lo stato di carica della propria batteria.
  - **RDM.2.3:** Il controller remoto deve visualizzare la velocità di rotazione dei motori del rover.
  - **RDM.2.4:** Il controller remoto deve visualizzare le distanze rilevate dai sensori a ultrasuoni anteriori.
  - **RDM.2.5:** Il controller remoto deve visualizzare la modalità di illuminazione abilitata.
  - **RDM.2.6:** Il controller remoto deve visualizzare la modalità di percezione degli ostacoli attualmente abilitata.
  - **RDM.2.7:** Il controller remoto deve visualizzare la modalità di dinamica di marcia abilitata.

- **RDM.2.8:** Il controller remoto deve visualizzare lo stato della supervisione del sistema.
- **RDM.2.9:** In assenza di una connessione attiva con la ricevente, il controller remoto deve indicare chiaramente all'operatore lo stato di non connessione.

## 2.2 Requisiti non funzionali

- **RNF.1:** Il sistema di supervisione e controllo deve essere realizzato tramite due Board STM32G4.
- **RNF.2:** L'accelerometro-giroscopio deve comunicare con la Board 2 tramite protocollo I2C.
- **RNF.3:** Il sistema di controllo remoto deve essere costituito da un modulo trasmittente e da un modulo ricevente.
  - **RNF.3.1:** Il modulo ricevente del controller remoto deve comunicare con la Board 2 tramite protocollo I2C.
- **RNF.4:** Il sistema deve garantire il rispetto di una deadline hard real-time per l'attivazione della frenata di emergenza, tale da impedire il contatto con un ostacolo rilevato lungo la direzione di marcia.

## **Parte II**

# **Stato e supervisione del sistema**

---

# Analisi degli aspetti critici

---

La stesura di questo capitolo parte dal concetto di *Dependability* (affidabilità), inteso come la capacità di un sistema di erogare un servizio *degno di fiducia* per tutta la durata operativa prevista, anche in presenza di guasti, disturbi e condizioni ambientali non ideali. In ambito embedded e, più in generale, nei sistemi cyber–fisici, questa proprietà non coincide con il semplice funzionare: un sistema può essere formalmente operativo e, allo stesso tempo, produrre comportamenti non sicuri o non controllabili. Per questo motivo, l'affidabilità viene affrontata come obiettivo progettuale trasversale, che include prevenzione del fallimento, tolleranza ai guasti e gestione controllata delle anomalie.

Nel rover tale esigenza è particolarmente rilevante perché il sistema integra:

- **attuazione fisica**, quindi capacità di generare effetti reali e potenzialmente pericolosi;
- **sensoristica e controllo** con dinamiche in tempo reale, dove le decisioni devono rispettare vincoli temporali e di correttezza per non degradare in comportamenti instabili;
- **architettura distribuita** su due board cooperanti (Master/Slave), in cui la correttezza globale dipende anche dalla comunicazione e dall'allineamento decisionale tra unità;
- **modalità operative di fallback** (*Degraded* e *Single-Board*) che devono garantire che, al degradare delle risorse disponibili, il sistema continui a comportarsi in modo prevedibile e sicuro.

L'obiettivo di questo capitolo è quindi duplice:

1. **identificare i top events tramite Fault Tree Analysis**, ossia le condizioni di sistema che configurano uno stato pericoloso, una perdita di controllo o una degradazione significativa del servizio;
2. **derivare, giustificare e formalizzare i tempi critici**, ovvero tutte quelle soglie temporali e fisiche che nel resto del documento sono state introdotte operativamente ma che necessitano di un'attenta analisi progettuale.

## NOTA

L'analisi dei top events è condotta esclusivamente sull'elettronica di potenza, controllo e sensoristica. I guasti software non vengono inclusi nei Fault Tree in quanto trattati tramite analisi statica, test e mitigati architetturalmente mediante supervisione, timeout e stati di fallback.

### 3.1 Fault Tree Analysis

Per ottenere un'analisi rigorosa, viene adottato il *Fault Tree Analysis* (FTA). Un fault tree consente di partire da un evento temuto e scomporlo logicamente in cause elementari e combinazioni di cause, tramite porte logiche (AND/OR) e meccanismi di dipendenza. In questo modo, non ci si limita a elencare possibili guasti, ma si costruisce una struttura causale che evidenzia:

- quali guasti sono *singolarmente sufficienti* a generare l'evento temuto (single point of failure);
- quali guasti diventano pericolosi *solo in combinazione*;

#### 3.1.1 FT1 - ROVER CRASH

Il Fault Tree FT1 analizza l'evento temuto P012 – ROVER CRASH. Questo albero non è associato a una specifica board, ma descrive un fallimento sistematico complessivo che può derivare da diverse catene causali.

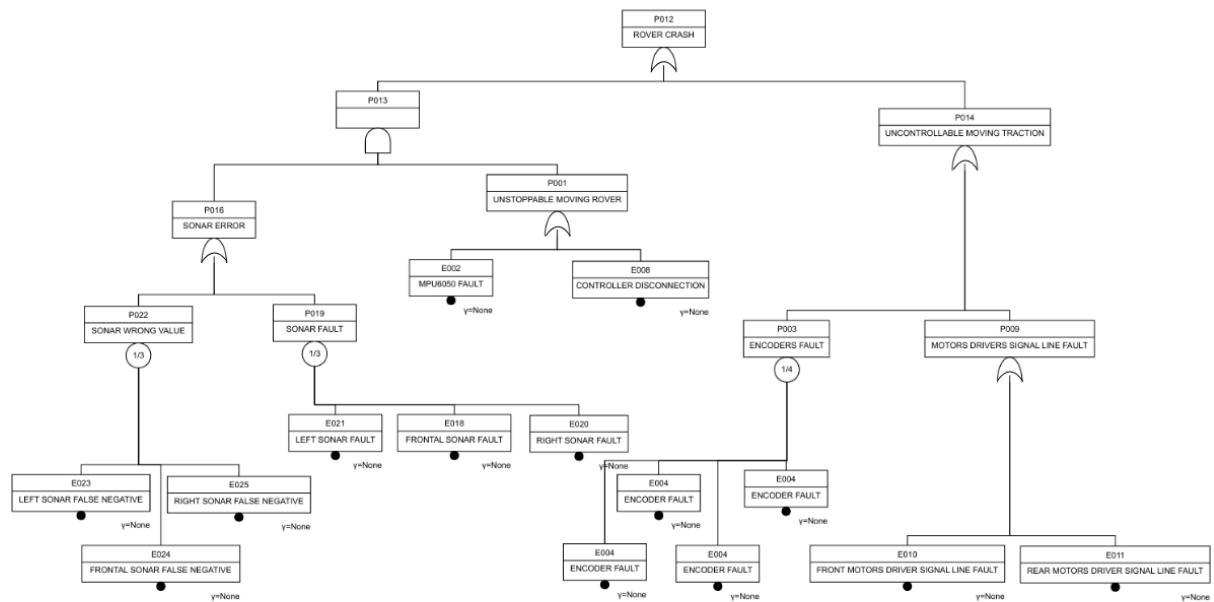


Figura 3.1: Fault Tree FT1 – Rover Crash

#### Livello 1 – Top Event: P012 – ROVER CRASH

Il nodo P012 rappresenta il top event dell'albero: *crash del rover*.

Dal diagramma, P012 è ottenuto come combinazione di due macro-scenari:

- un ramo sinistro (P013) che raccoglie condizioni che portano ad un comportamento pericoloso legato all'incapacità di **percezione dell'ambiente circostante** o all'incapacità di **recepire un comando di arresto**;
- un ramo destro (P014) che descrive condizioni di **trazione non controllabile** legate a feedback (encoder) e linee di comando (driver motori).

Il simbolo della porta sotto P012 è interpretabile come una porta logica di tipo OR.

## Livello 2A – Ramo sinistro: Percezione e Supervisione

Il nodo P013 aggrega due condizioni principali:

- P016 – SONAR ERROR
- P001 – UNSTOPPABLE MOVING ROVER

**P016 – SONAR ERROR** Questo nodo modella un malfunzionamento della catena sonar. È generato da due condizioni:

- P022 – SONAR WRONG VALUE
- P019 – SONAR FAULT

**P022 – SONAR WRONG VALUE (1/3)** Porta di voto **1/3**: l'evento si verifica se almeno uno dei tre sonar fornisce un valore errato (in particolare un falso negativo). In altre parole anche un singolo sonar che non rileva un ostacolo può compromettere la capacità di arresto. Eventi base:

- E023 – LEFT SONAR FALSE NEGATIVE
- E024 – FRONTAL SONAR FALSE NEGATIVE
- E025 – RIGHT SONAR FALSE NEGATIVE

**P019 – SONAR FAULT (1/3)** Porta di voto **1/3**: almeno uno dei tre sonar è in fault hardware: ciò equivale a un sensore rotto che non riesce a captare segnali di *ECHO*. Eventi base:

- E021 – LEFT SONAR FAULT
- E018 – FRONTAL SONAR FAULT
- E020 – RIGHT SONAR FAULT

**P001 – UNSTOPPABLE MOVING ROVER** Questo nodo rappresenta una condizione in cui il rover non può essere arrestato correttamente. Eventi base associati:

- E002 – MPU6050 FAULT
- E008 – CONTROLLER DISCONNECTION

Un fault dell'IMU o la perdita della connessione con il controller possono impedire una gestione coerente del moto.

## Livello 2B – Ramo destro: Trazione non controllabile

Il nodo: P014 – UNCONTROLLABLE MOVING TRACTION descrive una perdita del controllo della trazione.

È generato da:

- P003 – ENCODERS FAULT
- P009 – MOTORS DRIVERS SIGNAL LINE FAULT

**P003 – ENCODERS FAULT (1/4)** Porta di voto 1/4: almeno uno dei quattro encoder è in fault. La perdita anche di un singolo feedback può compromettere la regolazione della velocità e della traiettoria. Evento base:

- E004 – ENCODER anteriore sinistro in FAULT
- E004 – ENCODER anteriore destro in FAULT
- E004 – ENCODER posteriore sinistro in FAULT
- E004 – ENCODER posteriore destro in FAULT

**P009 – MOTORS DRIVERS SIGNAL LINE FAULT** Guasto sulle linee di comando verso i driver motore. Un guasto di linea può generare comandi incoerenti o incontrollati verso i motori. Eventi base:

- E010 – FRONT MOTORS DRIVER SIGNAL LINE FAULT
- E011 – REAR MOTORS DRIVER SIGNAL LINE FAULT

### 3.1.2 FT2 – ROVER UNABLE TO MOVE

Il Fault Tree riportato in Figura 3.2 analizza il Top Event: P001 – ROVER UNABLE TO MOVE. Questo evento rappresenta la condizione in cui il rover, pur essendo acceso, non è in grado di generare movimento. Rispetto al FT1 (Rover Crash), questo albero analizza una condizione di *fail-safe* (il rover non si muove), che pur non essendo pericolosa quanto il crash, rappresenta una perdita completa di servizio.

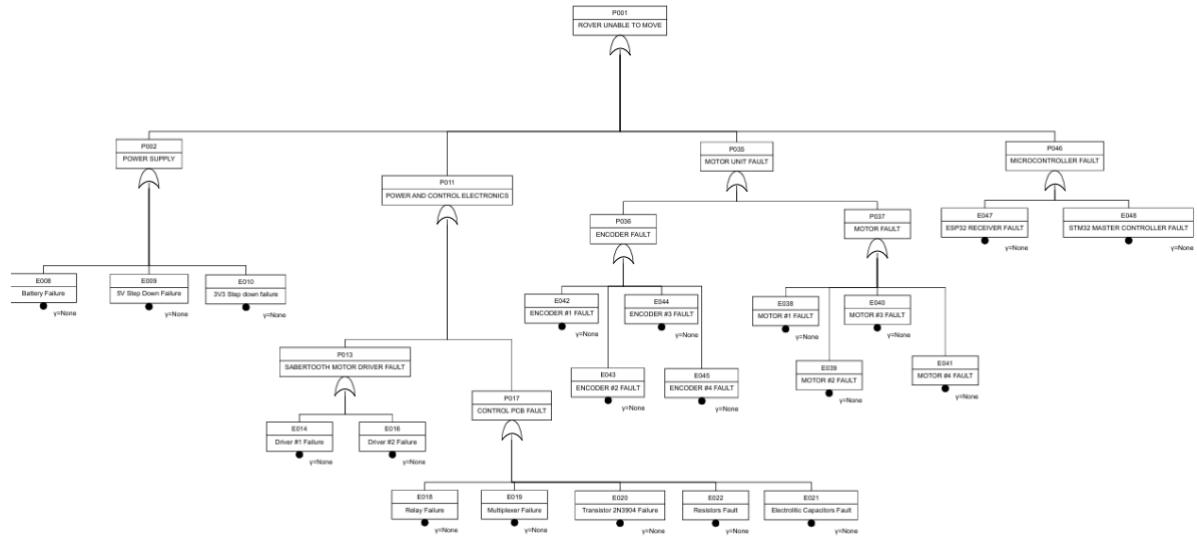


Figura 3.2: Fault Tree FT2 - Unable Move

### Livello 1 – Macro-cause

Il Top Event è generato da quattro macro-rami:

- P002 – POWER SUPPLY
- P011 – POWER AND CONTROL ELECTRONICS
- P035 – MOTOR UNIT FAULT

- P046 – MICROCONTROLLER FAULT

La porta logica in cima è ancora una OR: il verificarsi di una qualsiasi di queste condizioni rende il rover incapace di muoversi.

### Ramo 1 – Alimentazione: P002

Il nodo P002 – POWER SUPPLY rappresenta guasti nella catena di alimentazione. La perdita della batteria o dei regolatori DC/DC impedisce l'alimentazione di logica, driver e sensori. Eventi base:

- E008 – Battery Failure
- E009 – 5V Step Down Failure
- E010 – 3V3 Step Down Failure

### Ramo 2 – Power and Control Electronics: P011

Questo ramo aggrega i guasti nell'elettronica di controllo ed evidenzia come la PCB di interfaccia costituisca un punto critico nella catena di comando.

#### P013 – Sabertooth Motor Driver Fault Guasto del driver motori Sabertooth:

- E014 – Driver #1 Failure
- E016 – Driver #2 Failure

#### P017 – Control PCB Fault Guasto nella PCB di controllo:

- E018 – Relay Failure
- E019 – Multiplexer Failure
- E020 – Transistor 2N3904 Failure
- E022 – Resistors Fault
- E021 – Electrolytic Capacitors Fault

### Ramo 3 – Motor Unit Fault: P035

Il nodo P035 è causato da:

- P036 – Encoder Fault
- P037 – Motor Fault

esso mostra che il guasto anche di una singola unità motore o del relativo encoder può compromettere la capacità di movimento.

#### Encoder Fault Eventi base:

- E042 – Encoder anteriore sinistro in Fault
- E043 – Encoder anteriore destro in Fault
- E044 – Encoder posteriore sinistro in Fault
- E045 – Encoder posteriore destro in Fault

**Motor Fault** Eventi base:

- E038 - Motor anteriore sinistro in Fault
- E039 - Motor anteriore destro in Fault
- E040 - Motor posteriore sinistro in Fault
- E041 - Motor posteriore destro in Fault

#### Ramo 4 – Microcontroller Fault: P046

Il nodo P046 rappresenta il guasto di un microcontrollore nel sistema. In particolare la perdita della board master o del ricevitore rende impossibile generare segnali PWM validi verso i driver. Eventi base:

- E047 - ESP32 Receiver Fault
- E048 - STM32 Master Controller Fault

## 3.2 Strategie di mitigazione e gestione dei fault

In questa sezione vengono descritte le strategie progettuali adottate per mitigare i principali fault individuati mediante **Fault Tree Analysis**. L'obiettivo non è limitarsi a reagire al guasto una volta manifestato, ma definire a priori **meccanismi strutturati di rilevazione, isolamento e gestione controllata delle anomalie**, in modo da preservare, ove possibile, la continuità del servizio oppure garantire una **transizione sicura verso uno stato di arresto**.

Le contromisure illustrate derivano direttamente dai top events analizzati e sono organizzate secondo una logica funzionale: **architettura di controllo, catena di attuazione, sensoristica e condizioni energetiche e ambientali**. Per ciascun scenario vengono esplicitati il meccanismo di rilevazione, la strategia di mitigazione e lo stato operativo risultante (**modalità degradata o arresto fail-safe**), evidenziando le scelte architettoniche che consentono di ridurre l'impatto dei fault sul comportamento globale del rover.

### 3.2.1 Gestione dei fault delle board di controllo

L'architettura del rover è basata su una configurazione Master/Slave: la board Master detiene il controllo primario della catena di attuazione, mentre la board Slave fornisce supporto funzionale e retroazione. La gestione dei fault a livello di board è demandata congiuntamente al protocollo di comunicazione, alla logica di supervisione e a specifiche soluzioni hardware, quali il segnale di selezione dell'attuatore e il relè di potenza che alimenta i motor controller.

#### Failure della board Master

La perdita totale della board Master (blocco, spegnimento o reset non recuperabile) viene rilevata dalla board Slave tramite timeout del protocollo di comunicazione.

In tale condizione il segnale hardware di selezione viene automaticamente rilasciato, consentendo allo Slave di assumere il controllo logico della linea di comando. Tuttavia, non potendo garantire una gestione coordinata del moto, lo Slave forza un comando di arresto.

Poiché il relè di potenza che alimenta i motor controller è controllato direttamente dal Master, la sua perdita comporta la disaccoppiamento del relè e l'interruzione dell'alimentazione ai driver.

Lo stato risultante è l'arresto completo del rover, configurando una condizione di tipo **fail-safe**.

#### Failure della board Slave

La perdita della board Slave viene rilevata dal Master tramite timeout di comunicazione.

Il controllo della trazione rimane operativo sul Master; tuttavia, in assenza della retroazione fornita dallo Slave (encoder), non è più possibile garantire un controllo chiuso completo. Il sistema entra pertanto in **modalità degradata**, limitando la velocità massima a 80 rpm e adottando logiche conservative di controllo.

Lo stato risultante è un funzionamento con prestazioni ridotte ma comportamento ancora controllabile, configurando una **modalità degradata**.

### Failure simultanea di entrambe le board

La perdita funzionale di entrambe le board comporta l'assenza totale di controllo logico. Poiché il relè di potenza è controllato dal Master, la sua indisponibilità determina automaticamente la disalimentazione dei motor controller.

Lo stato risultante è l'interruzione dell'alimentazione alla catena di attuazione e l'arresto completo del rover, in configurazione **fail-safe**.

### 3.2.2 Gestione dei fault della catena di attuazione

La catena di attuazione del rover comprende motori DC, encoder e motor controller. A causa delle limitazioni dell'elettronica di bordo disponibile, non è possibile distinguere in modo deterministico la causa primaria di un malfunzionamento (encoder, driver o motore). È stato pertanto adottato un unico meccanismo di diagnostica funzionale, basato sull'analisi della coerenza tra comando e risposta dinamica complessiva del sistema.

Il sistema implementa un modulo di diagnostica predittiva basato su un'architettura a doppio buffer (Ping–Pong), che consente l'acquisizione continua dei dati senza interrompere l'elaborazione. Mentre un buffer registra in tempo reale i campioni di velocità provenienti dagli encoder, il secondo viene analizzato per calcolare l'*Integral Absolute Error* (IAE) tra il riferimento teorico e la velocità effettiva.

Il calcolo applica uno shift temporale che allinea il comando al tempo  $t$  con la risposta fisica al tempo  $t + \Delta t$ , compensando il ritardo dovuto all'inerzia meccanica. In questo modo vengono isolate esclusivamente discrepanze anomale rispetto al profilo dinamico atteso.

La Figura 3.3 mostra un esempio illustrativo del comportamento del sistema. A sinistra è riportato il confronto tra riferimento e velocità misurata, evidenziando il ritardo fisiologico e una condizione anomala. A destra è riportato il confronto tra l'IAE calcolato senza compensazione del ritardo e quello calcolato con shift temporale: si osserva come la compensazione riduca l'errore fisiologico e renda più evidente la presenza dell'anomalia.

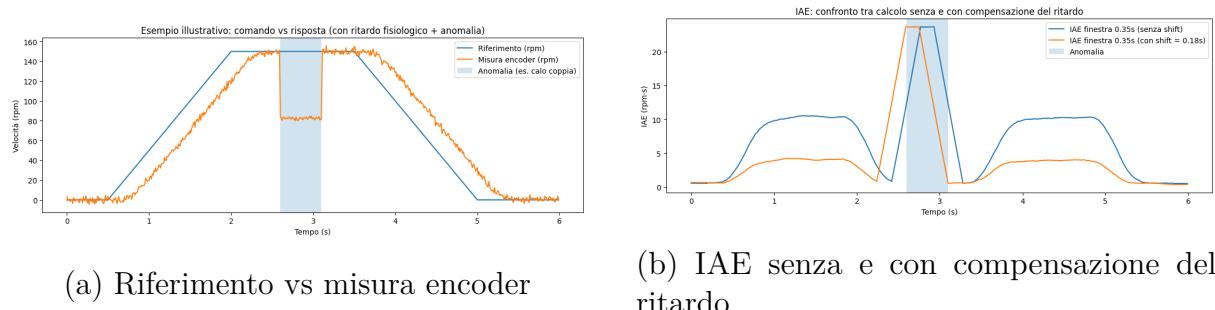


Figura 3.3: Esempio illustrativo del calcolo IAE con compensazione del ritardo fisiologico

Qualora l'errore integrale superi la soglia di tolleranza predefinita, lo stato del sistema viene aggiornato da **MOTOR\_HEALTHY** a **MOTOR\_FAILURE**. Alla rilevazione di tale condizione il sistema annulla immediatamente tutti i comandi di coppia, disabilita la catena di attuazione e forza la transizione allo stato di arresto controllato.

Non è prevista una modalità degradata per la trazione, poiché la perdita di coerenza dinamica anche su una singola unità può compromettere la stabilità del veicolo e generare comportamenti non controllabili. Lo stato risultante è pertanto l'arresto completo del rover, configurando una condizione di tipo **fail-safe**.

## NOTA

A causa delle limitazioni dell'elettronica di bordo disponibile (assenza di sensoristica dedicata e impossibilità di monitorare separatamente driver, motore ed encoder), non è possibile identificare con certezza la causa primaria del malfunzionamento. Il sistema adotta pertanto una diagnosi di tipo funzionale, basata esclusivamente sulla perdita di coerenza tra comando e risposta dinamica complessiva della trazione, senza identificare il componente fisico responsabile.

### 3.2.3 Gestione dei fault della sensoristica

La gestione dei fault relativi alla sensoristica è stata progettata considerando la natura fisica delle grandezze misurate, le modalità di interfacciamento (ADC o bus digitali) e le limitazioni diagnostiche intrinseche dei dispositivi impiegati.

#### Sensore di temperatura e misura della batteria

Il malfunzionamento della periferica ADC del microcontrollore o la presenza di incoerenze ripetute nelle misurazioni di temperatura e tensione di batteria viene gestito direttamente a livello di conversione analogico-digitale. Trattandosi di grandezze lentamente variabili, la diagnostica non si basa sul singolo campione ma su una verifica di persistenza dell'anomalia.

Il sistema memorizza il numero di errori consecutivi o conversioni non valide; al superamento di una soglia configurabile (nel sistema attuale pari a 5), il fault viene considerato persistente. In tale condizione viene inibita la modalità operativa attiva e il sistema transita nello stato di arresto controllato.

Lo stato risultante è l'arresto completo del rover, configurando una condizione di tipo fail-safe.

#### Accelerometro

Il malfunzionamento dell'IMU o la perdita di comunicazione con la periferica viene verificato sia in fase di inizializzazione sia durante il funzionamento normale. All'avvio, il microcontrollore effettua un reset dei registri del dispositivo e ne verifica la corretta configurazione; durante il funzionamento, i driver monitorano lo stato della periferica e consentono di rilevare eventuali errori di comunicazione.

Qualora l'inizializzazione o l'interrogazione della periferica non vadano a buon fine, il sistema può tentare un reset software del dispositivo. In caso di fault persistente, l'IMU viene considerata non affidabile.

In tale configurazione le rotazioni non vengono più basate sulla misura accelerometrica, ma vengono eseguite secondo un modello temporale deterministico, calcolato sulla base del tempo necessario a compiere una rotazione di 180°. Lo stato risultante è un funzionamento con precisione ridotta ma comportamento ancora controllabile, configurando una **modalità degradata**.

#### Controller remoto

La perdita di comunicazione con il controller remoto tramite bus I2C viene rilevata mediante timeout o errore segnalato dal driver di comunicazione.

Alla conferma della perdita del collegamento, i comandi di moto vengono annullati e il sistema transita nello stato di arresto controllato. Lo stato risultante è l'arresto completo del rover, configurando una condizione di tipo **fail-safe**.

### Sonar HC-SR04

Il malfunzionamento del sensore ultrasonico viene rilevato confrontando il tempo di echo misurato con il limite temporale massimo previsto dal dispositivo. Poiché per costruzione il sensore genera comunque un segnale entro un intervallo definito, la mancata risposta entro tale finestra viene interpretata come anomalia.

In caso di mancata risposta, alla distanza corrispondente viene associato un valore conservativo di sicurezza, mantenendo coerenza con la logica di arresto e assumendo la presenza di un ostacolo entro una distanza cautelativa. Lo stato risultante è un funzionamento conservativo, configurando una **modalità degradata**.

#### Problematiche legate ai sonar HC-SR04

L'analisi dei Fault Tree ha evidenziato come la catena sonar contribuisca direttamente ai Top Events, rendendo necessaria una riflessione specifica sulle caratteristiche dei sensori impiegati.

Gli HC-SR04 sono dispositivi ultrasonici a basso costo, privi di diagnostica interna e di meccanismi di autoverifica. In particolare:

- non forniscono informazioni strutturate sul proprio stato di salute;
- possono generare falsi negativi in presenza di superfici inclinate o materiali poco riflettenti;
- sono sensibili a interferenze e fenomeni di cross-talk in configurazioni multi-sensore.

A causa di tali limitazioni strutturali, la strategia teoricamente prevista per la gestione del fault — basata su una discriminazione affidabile tra misura valida, assenza reale di ostacolo e guasto del sensore — **non è stata implementata nella versione effettiva del sistema**.

Le caratteristiche costruttive del dispositivo non consentono infatti di distinguere in modo deterministico tra:

assenza reale di ostacolo e misura non affidabile o failure del sensore.

Inoltre, non è stato possibile introdurre tecniche di filtraggio o media temporale delle misure per migliorare l'affidabilità della stima, poiché il sistema opera con vincoli temporali stringenti necessari a garantire il corretto funzionamento della supervisione e delle logiche di arresto. L'introduzione di finestre di media o filtri più complessi avrebbe aumentato la latenza decisionale, compromettendo la reattività complessiva del sistema.

Ne consegue che non è stato possibile integrare una politica di fallback strutturata fondata esclusivamente sulla catena ultrasonica.

### 3.2.4 Condizioni energetiche e termiche degradanti

Le seguenti condizioni non comportano un arresto immediato del sistema, ma determinano una transizione verso modalità operativa degradata, con limitazione delle prestazioni al fine di preservare sicurezza, stabilità termica e continuità del servizio.

#### Controller remoto scarico

La riduzione del livello di carica del controller remoto, tale da comprometterne l'autonomia operativa, determina l'ingresso del sistema in modalità degradata. In questa configurazione vengono limitate la velocità massima, le accelerazioni consentite e le eventuali modalità operative ad alte prestazioni.

L'obiettivo è ridurre il consumo energetico complessivo e consentire una gestione controllata del sistema fino allo spegnimento o al ripristino della carica. Lo stato risultante è un funzionamento con prestazioni ridotte ma comportamento ancora controllabile, configurando una modalità degradata.

#### Temperatura interna oltre soglia

Il superamento di una soglia di temperatura interna, mantenuto per un intervallo temporale minimo definito progettualmente (specificato successivamente), viene rilevato tramite monitoraggio continuo del sensore analogico. La verifica di persistenza consente di evitare falsi positivi dovuti a transienti termici.

In tale condizione il sistema entra in modalità degradata, riducendo la potenza erogata, limitando la velocità massima ed escludendo modalità ad alto assorbimento energetico. Lo stato risultante è un funzionamento conservativo volto a prevenire condizioni di surriscaldamento critico, configurando una modalità degradata.

#### Batteria sotto il 23% di carica

Il livello di carica della batteria principale viene stimato tramite misura ADC della tensione e confronto con la soglia impostata al 23%. Al di sotto di tale valore il sistema transita in modalità degradata.

In questa configurazione vengono limitate la velocità massima e le modalità ad alte prestazioni, adottando una gestione conservativa del consumo energetico al fine di prolungare l'autonomia residua e garantire una conclusione controllata dell'operatività. Lo stato risultante è un funzionamento con capacità ridotte ma ancora controllabile, configurando una modalità degradata.

### 3.2.5 Sintesi complessiva delle politiche di gestione dei fault

Di seguito è riportata la sintesi complessiva delle politiche di gestione dei fault descritte nella presente sezione. La tabella riassume, per ciascun evento significativo, la strategia di mitigazione adottata e lo stato operativo risultante.

Fault	Azione di mitigazione	Stato risultante
Failure Master	Disalimentazione dei driver di potenza e arresto del sistema	Arresto (fail-safe)
Failure Slave	Limitazione della velocità massima a 80 rpm e controllo conservativo	Modalità degradata
Failure Master + Slave	Disalimentazione automatica della catena di attuazione	Arresto (fail-safe)
Fault trazione (IAE)	Stop immediato e disabilitazione completa dell'attuazione	Arresto (fail-safe)
Fault ADC persistente	Transizione allo stato di arresto controllato	Arresto (fail-safe)
IMU non disponibile	Rotazione temporizzata non basata su retroazione accelerometrica	Modalità degradata
Perdita controller remoto	Annullamento dei comandi di moto	Arresto (fail-safe)
Sonar non affidabile	Assegnazione distanza conservativa e comportamento prudenziale	Modalità degradata
Controller remoto scarico	Limitazione delle prestazioni e riduzione dei consumi	Modalità degradata
Temperatura oltre soglia	Riduzione della potenza erogata e limitazione delle prestazioni	Modalità degradata
Batteria < 23%	Limitazione delle prestazioni e gestione conservativa dell'energia	Modalità degradata

Tabella 3.1: Sintesi delle strategie di mitigazione e gestione dei fault

### 3.3 Definizione e giustificazione dei tempi critici

Dopo aver identificato i principali *top events* tramite Fault Tree Analysis, è necessario giustificare formalmente le soglie temporali e fisiche che nel progetto sono state introdotte come parametri operativi. Tali soglie non sono arbitrarie, ma derivano da vincoli fisici, specifiche dei componenti e considerazioni di sicurezza sistemica.

#### 3.3.1 Protezione Termica – Temperatura Massima Ammissibile

La strategia di protezione termica è stata dimensionata considerando:

- le specifiche del microcontrollore STM32G474RE;
- i vincoli di dissipazione del rover;
- la necessità di evitare interventi inutilmente conservativi.

Il microcontrollore garantisce operatività fino a una temperatura di giunzione massima:

$$T_J^{max} = 130^{\circ}C$$

Tuttavia, progettare il sistema per operare in prossimità del limite assoluto comporterebbe una riduzione dell'affidabilità a lungo termine oltre che aumento del rischio di fault intermittenti. Per questo motivo è stata introdotta una soglia preventiva pari a:

$$T_{crit} = 100^{\circ}C$$

che garantisce un margine di sicurezza di:

$$\Delta T = 30^{\circ}C$$

Al fine di evitare falsi positivi dovuti a rumore sensoriale o transitori termici irrilevanti, la temperatura deve permanere sopra la soglia critica per un periodo continuativo di **10 secondi**. Solo a quel punto il sistema riduce la velocità massima, permettendo la dissipazione del calore in eccesso senza interrompere il funzionamento.

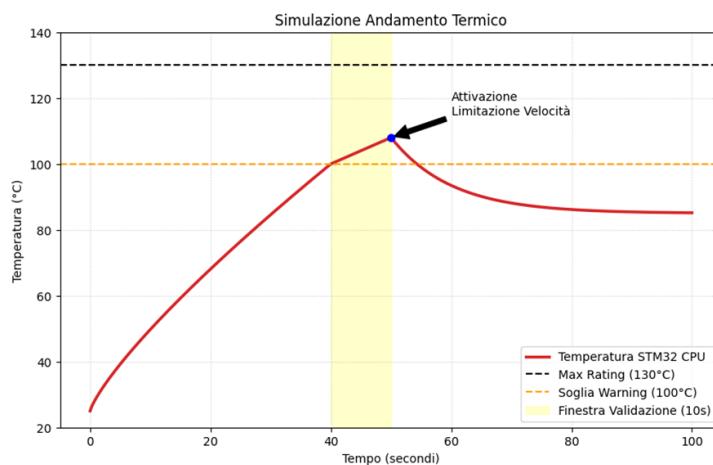


Figura 3.4: Simulazione andamento termico

### 3.3.2 Discriminazione Ostacolo Statico / Ostacolo in Movimento

Per discriminare un ostacolo in movimento da uno statico è stato necessario definire un intervallo temporale massimo  $\Delta t$  entro cui lo stesso ostacolo deve essere rilevato consecutivamente da due sensori a ultrasuoni differenti.

#### Analisi cinematica nel caso peggiore

Si considera il caso peggiore:

- distanza iniziale massima: 3 m;
- angolo rispetto all'asse di marcia: 45°.

La proiezione lungo l'asse di marcia è:

$$d_{\text{proiezione}} = 3 \cdot \cos(45^\circ) \approx 2.12 \text{ m}$$

Affinché lo stesso ostacolo venga rilevato dal sensore centrale, il rover deve percorrere:

$$d_{\text{necessaria}} = 3 - 2.12 \approx 0.88 \text{ m}$$

Considerando la velocità massima del rover:

$$v_{\max} = 1.24 \text{ m/s}$$

il tempo minimo fisicamente plausibile risulta:

$$t_{\min} = \frac{0.88}{1.24} \approx 0.71 \text{ s}$$

Il valore teorico di 0.71 s rappresenta una stima ideale. Tenendo conto delle incertezze dovute al rumore dei sensori a ultrasuoni, alla discretizzazione temporale delle misure e alle latenze di elaborazione, è stato introdotto un margine di sicurezza. Il valore finale scelto per l'intervallo temporale è quindi pari a  $\Delta t = 1 \text{ s}$ . Superato tale intervallo, l'ostacolo viene considerato statico

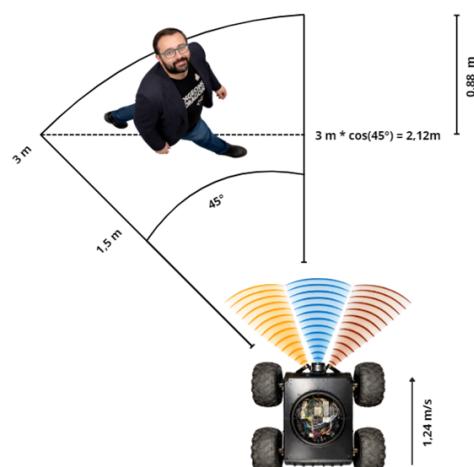


Figura 3.5: Analisi cinematica prof in movimento

---

# Progettazione della supervisione

---

La presente sezione ha l’obiettivo di dettagliare il sottosistema di supervisione e controllo progettato per il rover, ponendo l’accento sulle metodologie adottate per assicurare l’affidabilità e la sincronizzazione tra le unità di elaborazione distribuite. L’architettura è stata concepita non solo per garantire le prestazioni in condizioni operative nominali, ma soprattutto per gestire in sicurezza eventuali criticità, implementando strategie di degradazione controllata qualora si verifichino malfunzionamenti nel canale di comunicazione o nelle singole unità. A tale scopo, è stato adottato un paradigma architettonico **Master-Slave asincrono**, supportato da un protocollo di comunicazione robusto che prevede lo scambio iterativo e validato di tre gruppi di informazioni fondamentali: *stato locale*, *stato globale* e *decisione di attuazione*. La consistenza dei dati scambiati è garantita da meccanismi di controllo d’integrità (CRC), gestione delle conferme e politiche di ritrasmissione. Nello specifico, il ruolo di **Master** è assegnato alla **Board 2**, in quanto si interfaccia direttamente con la logica di comando di alto livello, mentre la **Board 1** opera come nodo **Slave**. L’architettura di controllo prevede una gestione del comportamento del rover articolata in tre **modalità di funzionamento** principali, ciascuna progettata per rispondere a specifici scenari di operatività e affidabilità.

## 4.1 Modalità normale

In condizioni operative nominali, il sistema garantisce la piena operatività di entrambe le unità di elaborazione e l’integrità del canale di comunicazione. Il protocollo di supervisione è strutturato su un ciclo di controllo iterativo suddiviso in tre macro-fasi sequenziali:

1. **Scambio dello Stato Locale**: Scambio bidirezionale dei dati sensoriali acquisiti dalle singole board.
2. **Scambio dello Stato Globale**: Costruzione e scambio di una visione globale del sistema.
3. **Scambio della Decisione**: Calcolo distribuito e scambio della decisione di attuazione.

Per garantire il determinismo temporale necessario in un sistema *safety-critical*, è stata implementata una politica di gestione degli errori di tipo *fail-fast*: è consentita al massimo **una ritrasmissione per board** (`max_rtrasmissioni_per_board = 1`) all’interno di ogni ciclo. Questa scelta progettuale previene latenze eccessive dovute a tentativi multipli di recupero su un canale instabile, permettendo al sistema di identificare rapidamente condizioni di guasto persistente e transire verso stati di sicurezza.

Inizialmente, il protocollo prevede una fase di **avvio della sessione**, propedeutica allo scambio dati e necessaria per stabilire il sincronismo tra le due unità. La procedura viene avviata dalla **Board 2 (Master)**, che asserisce i segnali di controllo hardware: imposta `SESSION = 1` per marcare l’inizio di un nuovo ciclo di supervisione e porta `M_TALK = 1` per segnalare la propria intenzione di trasmettere informazioni. Alla rilevazione di questi segnali, la **Board 1 (Slave)** si attiva e notifica la propria disponibilità all’ascolto manipolando la propria linea hardware di handshake (abbassando il segnale `S_TALK`), completando così la sincronizzazione iniziale.

Il protocollo di scambio delle informazioni è poi strutturato in tre macro-fasi successive, di seguito analizzate nel dettaglio:

#### 4.1.1 Scambio dello Stato Locale

Una volta stabilita la connessione, avviene lo scambio bidirezionale delle informazioni sensoriali (stato locale). Questa fase segue una sequenza rigorosa che sfrutta le transizioni delle linee hardware di talk come meccanismo di conferma (ACK) implicito, garantendo la sincronizzazione senza l'overhead di messaggi di controllo aggiuntivi. La procedura ha inizio con la **Board 2 (Master)**, che trasmette il proprio pacchetto contenente lo stato locale e il relativo checksum (CRC). Se avviene la ricezione, la **Board 1 (Slave)** verifica l'integrità dei dati:

- **In caso di successo (CRC valido):** La Board 1 conferma la ricezione portando a livello logico alto la propria linea di segnalazione ( $S\_TALK = 1$ ). Questa singola transizione assume una duplice valenza: funge da **ACK implicito** per i dati del Master e, simultaneamente, notifica che lo Slave è pronto a trasmettere il proprio stato.
- **In caso di errore (CRC errato):** Il sistema verifica il contatore delle ritrasmissioni ( $B2\_ritrasmisioni$ ). Essendo il limite fissato a 1, è consentito un unico tentativo di reinvio. Qualora anche la ritrasmissione fallisca, o scada il timeout di attesa, la sessione viene immediatamente interrotta ed entrambe le schede transitano nello stato **DEGRADATO**.

Rilevato l'ACK implicito (innalzamento di  $S\_TALK$ ), la Board 2 si predispone alla ricezione abbassando la propria linea di richiesta ( $M_TALK = 0$ ). A questo punto, la Board 1 trasmette il proprio pacchetto (stato locale + CRC) e la Board 2, se lo riceve, ne verifica la correttezza:

- **In caso di successo:** Non è previsto l'invio di un ACK esplicito; la validazione della ricezione è sancita dal passaggio automatico del Master alla fase successiva (**Fase 3: Scambio dello stato globale**).
- **In caso di errore:** Si applica la medesima politica di sicurezza descritta in precedenza ( $B1\_ritrasmisioni < 1$ ). Il fallimento della ritrasmissione da parte dello Slave determina l'immediata transizione del sistema allo stato **DEGRADATO**, costringendo le unità a operare in modalità indipendente per preservare la sicurezza del veicolo.

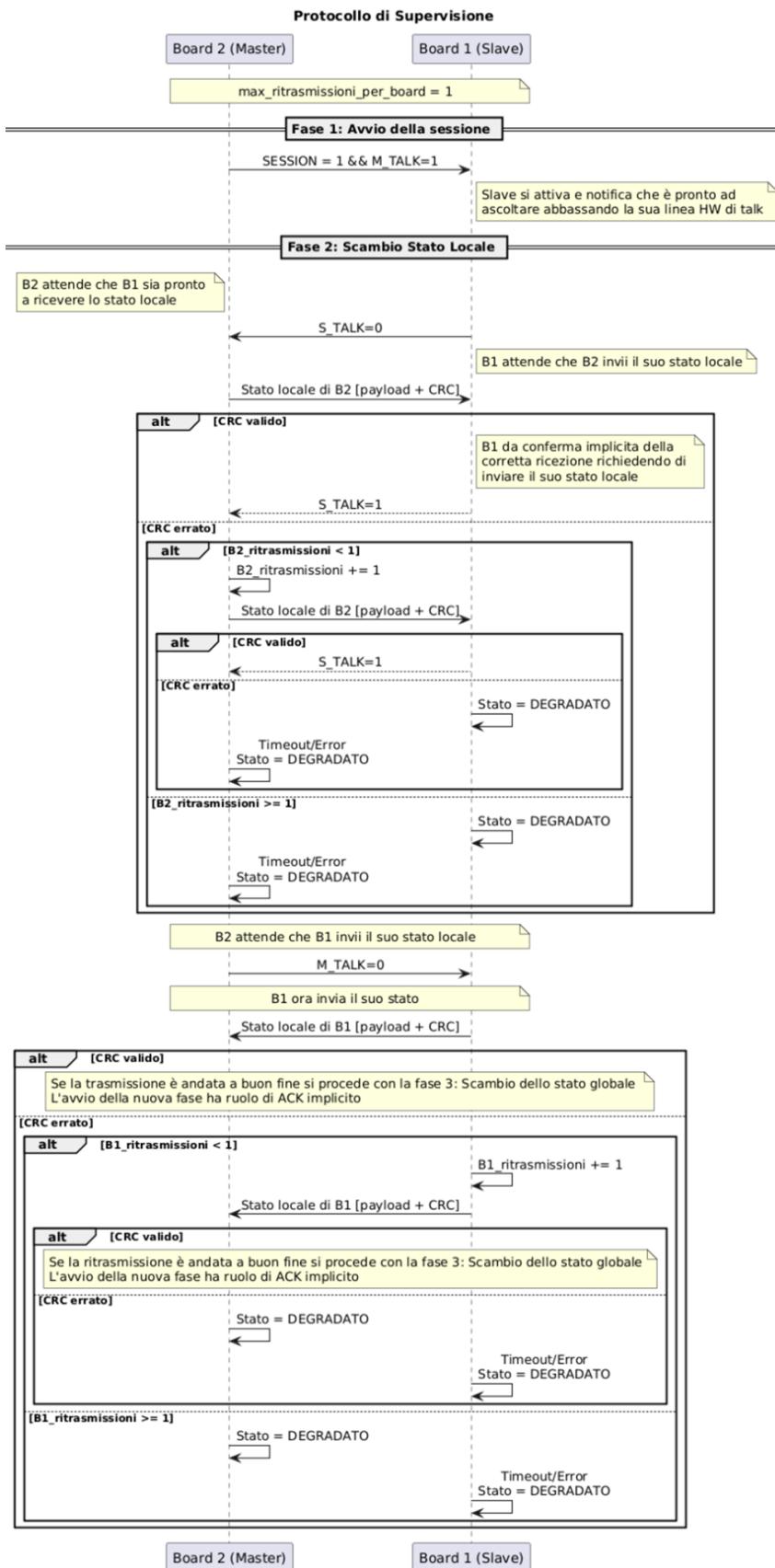


Figura 4.1: Scambio dello Stato Locale

#### 4.1.2 Scambio dello Stato Globale

Al termine dell'acquisizione dei dati locali della controparte, ciascuna unità procede autonomamente all'aggregazione delle informazioni, generando una visione unificata del sistema. Su questa base dati vengono poi calcolate le variabili derivate necessarie al controllo. Per garantire la coerenza operativa, è imperativo che entrambe le schede dispongano della medesima "fotografia" del sistema prima di deliberare l'azione di controllo. A tal fine, il protocollo avanza alla fase di scambio e verifica dei rispettivi stati globali calcolati. La procedura segue una logica speculare alla fase precedente, in particolare: La Board 2 (Master) avvia la nuova fase asserendo  $M\_TALK = 1$ . Questa transizione assume una duplice valenza critica: segna l'inizio dello scambio globale e, contestualmente, funge da **ACK implicito** per la corretta ricezione dello stato locale dello Slave inviato nella fase precedente. In risposta, la Board 1 (Slave) si predisponde alla ricezione portando  $S\_TALK = 0$ , completando così l'allineamento iniziale. Successivamente, la Board 2 invia il proprio pacchetto contenente lo Stato Globale. La Board 1 se riceve i dati, ne valida l'integrità tramite CRC:

- **Esito Positivo:** La Board 1 alza la linea  $S\_TALK = 1$ . Come avvenuto in precedenza, questo segnale agisce sia da conferma di ricezione sia da notifica di prontezza all'invio dei propri dati.
- **Esito Negativo:** In presenza di errore (CRC invalido), il sistema tenta una singola ritrasmissione ( $B2\_ritrasmissioni < 1$ ). Se l'errore persiste, il ciclo viene abortito e le schede transitano immediatamente nello stato **DEGRADATO**.

Ricevuta la conferma dallo Slave, la Board 2 cede il bus abbassando la linea  $M\_TALK = 0$ . La Board 1 trasmette il proprio Stato Globale verso il Master, che, se lo riceve, ne verifica l'integrità:

- **Esito Positivo:** La validazione corretta da parte del Master conclude la fase di allineamento. Il sistema procede quindi direttamente alla **Fase 4: Scambio della decisione**.
- **Esito Negativo:** Analogamente al caso precedente, viene effettuato un tentativo di ritrasmissione. Il fallimento definitivo dello scambio costringe il sistema ad abbandonare la modalità cooperativa, portando entrambe le unità nello stato **DEGRADATO** per gestire la sicurezza in autonomia.

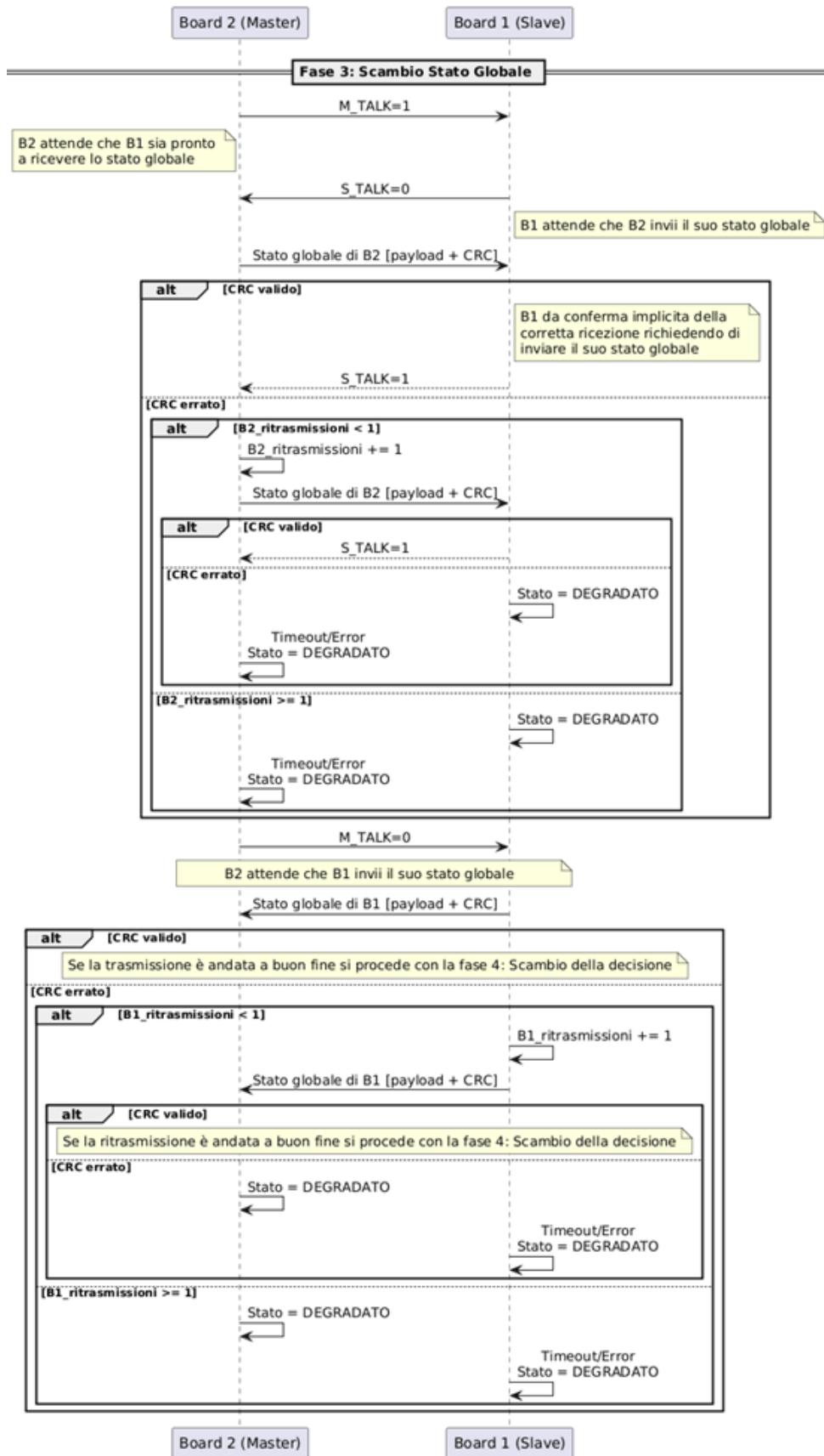


Figura 4.2: Scambio dello Stato Globale

### 4.1.3 Scambio della Decisione

Disponendo ora di una base di conoscenza unificata (Stato Globale), ciascuna unità elabora parallelamente l'algoritmo di controllo per determinare la Decisione di Attuazione. La Board 2 (Master) inaugura la nuova fase asserendo  $M\_TALK = 1$ . In questo stadio preliminare, prima ancora di avviare la trasmissione, viene effettuata una verifica interna critica sulla coerenza degli stati globali precedentemente scambiati: qualora le visioni del sistema risultino discordanti, il protocollo interrompe immediatamente la cooperazione e le unità commutano in modalità Single-board per gestire il veicolo tramite procedura locale. Se invece la congruenza è confermata, la Board 1 (Slave) si predispone alla ricezione portando  $S\_TALK = 0$ . A canale stabilito, la Board 2 invia il pacchetto contenente la propria decisione operativa. La Board 1, se riceve i dati, ne valida l'integrità tramite CRC:

- **Esito Positivo:** La Board 1 alza la linea  $S\_TALK = 1$ . Analogamente alle fasi precedenti, questo segnale costituisce l'ACK implicito per la decisione del Master e segnala la prontezza dello Slave a inviare la propria deliberazione.
- **Esito Negativo:** In caso di errore (CRC invalido) o mancata ricezione, il sistema tenta una singola ritrasmissione ( $B2\_ritrasmisioni < 1$ ). Se l'anomalia persiste, il ciclo viene abortito e le schede transitano immediatamente nello stato **DEGRADATO**.

Acquisita la conferma, la Board 2 cede il controllo del bus portando  $M\_TALK = 0$ . La Board 1 trasmette la propria decisione verso il Master, che, se riceve il pacchetto, ne verifica l'integrità:

- **Esito Positivo:** La validazione corretta da parte del Master conclude lo scambio dati. Il sistema avanza quindi alla fase di **chiusura della sessione**, formalizzata dal Master che porta a livello basso la linea SESSION = 0. È in questo stadio che avviene il confronto finale: le unità verificano la corrispondenza tra la decisione locale e quella ricevuta. Qualora esse risultino **divergenti**, l'incoerenza viene trattata come guasto critico, costringendo entrambe le unità a transitare immediatamente in modalità **Single-board** per operare in autonomia; in caso contrario, il consenso è raggiunto e la comunicazione termina.
- **Esito Negativo:** Come da prassi, un eventuale errore innesca il tentativo di ritrasmissione. Il fallimento definitivo dello scambio costringe il sistema ad abbandonare la modalità cooperativa, portando entrambe le unità nello stato **DEGRADATO** per garantire la sicurezza in autonomia.

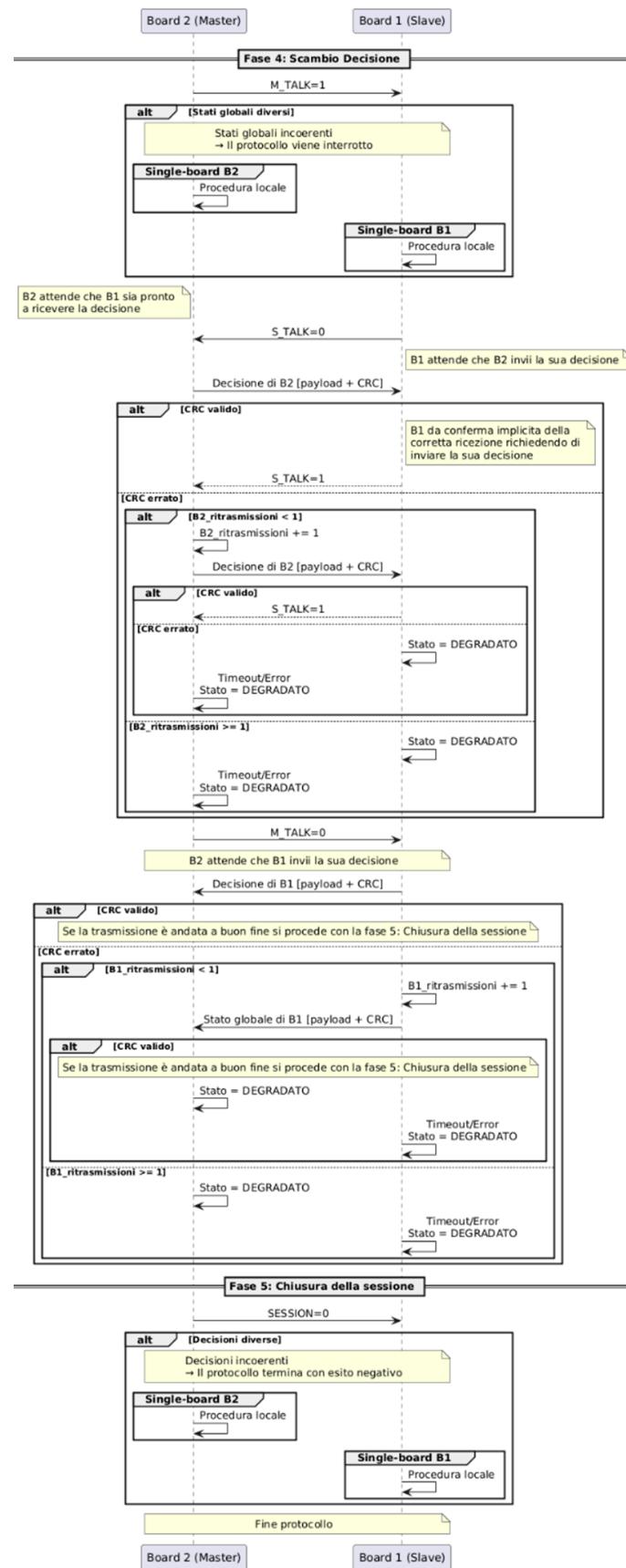


Figura 4.3: Scambio decisione

## 4.2 Modalità degradata

La **modalità degradata** rappresenta una condizione intermedia di funzionamento in cui **almeno una delle due board è ancora operativa**, ma la comunicazione risulta **inaffidabile o parzialmente compromessa**. Questa modalità viene attivata quando il protocollo di comunicazione rileva errori persistenti che non possono essere risolti tramite i normali meccanismi di ritrasmissione. Di conseguenza, ciascuna board interrompe il coordinamento e **procede in modo autonomo**, adottando una **scelta di attuazione indipendente** basata esclusivamente sulle informazioni locali disponibili. Successivamente, entrambe le board tentano di verificare il ripristino delle linee di comunicazione precedentemente interrotte o non funzionanti. Quindi, le board avviano uno scambio di messaggi che replica strutturalmente le tre fasi del ciclo nominale (scambio stati e decisione). Tuttavia, in questa fase, i payload operativi vengono sostituiti da **pacchetti di PING** (contenenti un byte di *dummy data*) così da permettere la verifica della correttezza della sequenza di handshake e la stabilità del collegamento. Il completamento con successo dell'intera sequenza di test (validazione bidirezionale tramite PING) sancisce il formale ripristino dell'integrità del canale di comunicazione. In tale scenario, a partire dal ciclo di controllo immediatamente successivo, il sistema innesca la transizione di stato verso la modalità **Normale**: le unità abbandonano la logica di fallback e riattivano il protocollo di supervisione cooperativo distribuito, riallineando la base di conoscenza condivisa. Viceversa, qualora la procedura di diagnostica evidenzi la persistenza di instabilità o la perdita di pacchetti, il sistema conferma la permanenza nello stato **Degradoato**. Le board reiterano l'esecuzione autonoma e disaccoppiata, rinviando il tentativo di riaggancio al ciclo di verifica successivo.

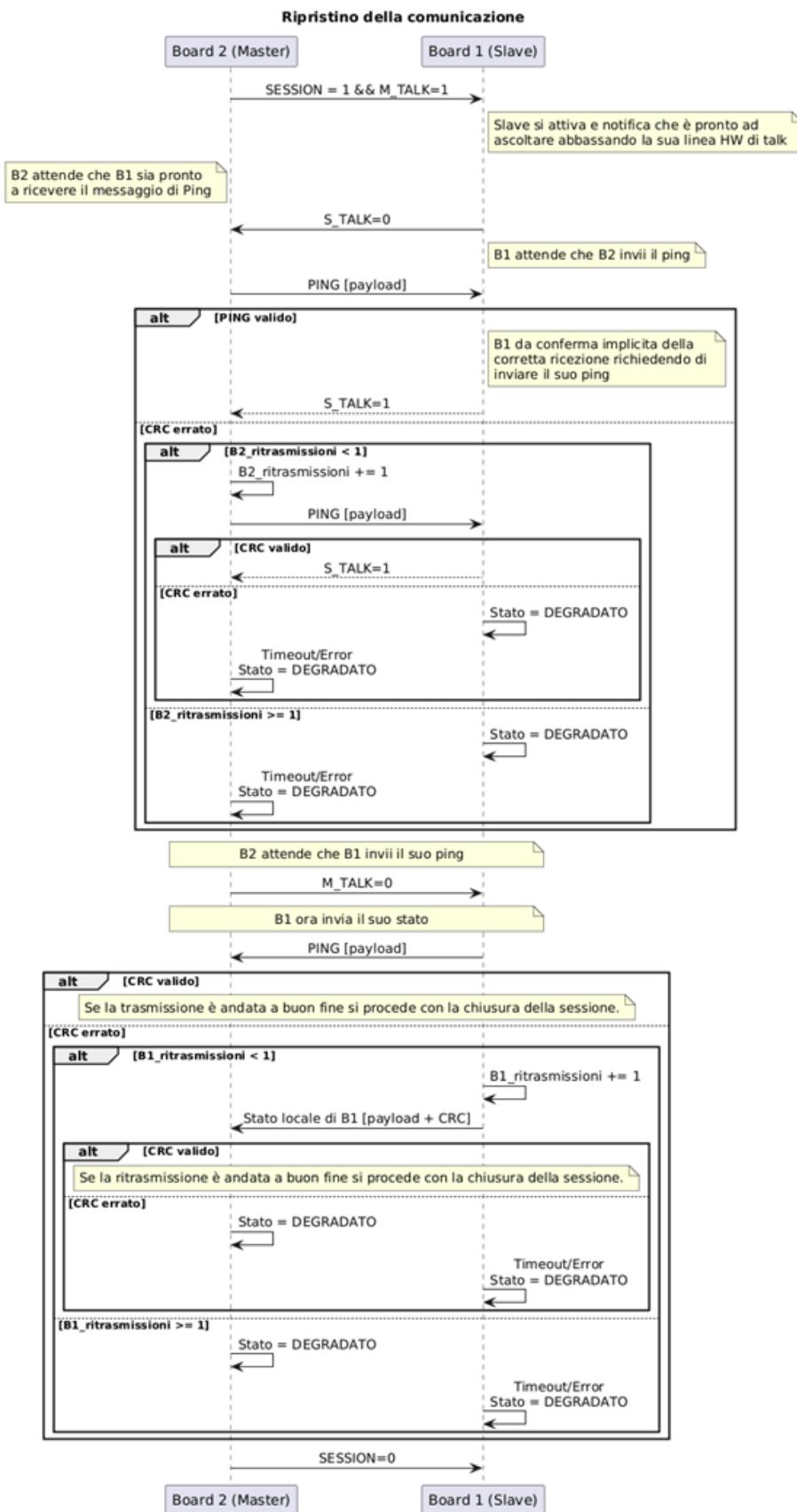


Figura 4.4: Protocollo in Stato Degradato

### 4.3 Modalità single-board

Questa modalità rappresenta lo stato di emergenza irreversibile del sistema, attivato esclusivamente qualora i meccanismi di validazione rilevino una **divergenza** nello Stato Globale o nella Decisione di Attuazione. Tale incongruenza è considerata sintomatica di un grave malfunzionamento logico o hardware in almeno una delle unità di elaborazione, rendendo di fatto inaffidabile qualsiasi ulteriore tentativo di cooperazione. In risposta a tale evento, il sistema applica una strategia di **confinamento del guasto**. Ogni unità procede quindi in isolamento funzionale, eseguendo le proprie routine di controllo locali per portare il rover in una condizione di sicurezza o arresto. A differenza della modalità Degradata, lo stato Single-Board non prevede alcuna procedura software di ripristino automatico. L'uscita da questo stato e il ripristino delle condizioni nominali sono possibili unicamente attraverso un **Hard Reset** del sistema (riavvio fisico delle board), necessario per reinizializzare le periferiche.

---

# Definizione delle variabili di stato

---

Il presente capitolo descrive le variabili di stato impiegate dalla logica di supervisione del rover, specificandone organizzazione e tipizzazione all'interno del modello.

La componente di supervisione è stata sviluppata secondo un approccio **Model-Based Design (MBD)**, utilizzando **Simulink** e **Stateflow** per la modellazione della macchina a stati e delle logiche decisionali di alto livello. Stateflow, formalmente affine ai diagrammi di stato gerarchici (Statecharts), consente di rappresentare in modo strutturato il comportamento operativo del sistema.

Nel modello di supervisione, le variabili di stato costituiscono le informazioni persistenti che descrivono la configurazione dinamica del rover a livello decisionale e che influenzano:

- le transizioni tra stati;
- le condizioni di attivazione;
- le azioni associate agli stati della macchina di controllo.

Le funzionalità di basso livello (driver hardware, gestione delle periferiche e controllo locale) sono implementate direttamente nel firmware. La logica di supervisione è invece modellata integralmente in ambiente Simulink/Stateflow, dove le variabili vengono definite tramite enumerazioni, oggetti **Simulink.Bus** e tipi primitivi MATLAB, in modo coerente con l'architettura del modello.

Nel seguito del capitolo le variabili sono presentate secondo la loro definizione nel modello, evidenziandone ruolo funzionale, struttura e tipizzazione. Gli aspetti relativi alla rappresentazione nel codice C generato e alla loro traduzione nel firmware target saranno trattati nel capitolo dedicato alla generazione del codice.

## 5.1 Enumerazioni

Le enumerazioni definiscono insiemi finiti di valori discreti utilizzati dalla logica di supervisione per rappresentare modalità operative, condizioni di sistema e configurazioni di segnalazione.

Nel modello Stateflow esse sono implementate come classi derivate da **Simulink.IntEnumType**, garantendo tipizzazione forte e coerenza nella generazione automatica del codice C. Ogni valore enumerativo è associato a un codice intero, che ne costituisce la rappresentazione effettiva nel codice generato.

Le enumerazioni sono organizzate secondo il dominio funzionale di appartenenza, distinguendo tra:

- controllo e dinamica del rover;
- segnalazione visiva;
- diagnostica e percezione dell'ambiente.

### 5.1.1 Controllo e dinamica

#### BRAKING\_TYPE

L'enumerazione BRAKING\_TYPE definisce le modalità di arresto applicabili ai motori del rover. Essa viene utilizzata dalla logica decisionale per determinare il comportamento del sistema in fase di decelerazione o arresto.

Nome	Valore	Descrizione
NONE	0	Nessuna frenata attiva.
NORMAL	1	Frenata progressiva controllata.
EMERGENCY	2	Frenata di emergenza con arresto rapido dei motori.

Tabella 5.1: Enumerazione BRAKING\_TYPE

#### ROVER\_MODE

L'enumerazione ROVER\_MODE seleziona il profilo di guida del rover, influenzando la mappatura della potenza e la risposta dinamica del sistema.

Nome	Valore	Descrizione
DEFAULT	0	Mappatura standard, bilanciata tra prestazioni e consumo energetico.
SPORT	1	Profilo ad alte prestazioni con accelerazione più reattiva.
ECO	2	Profilo a risparmio energetico con accelerazione progressiva.

Tabella 5.2: Enumerazione ROVER\_MODE

### 5.1.2 Segnalazione visiva

#### LED\_TYPE

L'enumerazione LED\_TYPE definisce i pattern di illuminazione dei LED frontali del rover, utilizzati per comunicare lo stato operativo verso l'esterno.

Nome	Valore	Descrizione
OFF	0	LED spenti.
WHITE	1	Luce fissa di colore bianco.
RED	2	Luce fissa di colore rosso.
BLINKING_WHITE	3	Lampeggio di colore bianco.
BLINKING_RED	4	Lampeggio di colore rosso.
BLINKING_RED_SLOW	5	Lampeggio rosso lento.

Tabella 5.3: Enumerazione LED\_TYPE

**REAR\_LED\_TYPE**

L'enumerazione REAR\_LED\_TYPE definisce le modalità di funzionamento della fanaleria posteriore del rover.

Nome	Valore	Descrizione
IDLE	0	Stato di riposo.
BACKLIGHTS	1	Luci di posizione posteriori (rosse fisse).
BRAKING_LIGHTS	2	Luci di stop attive durante la frenata.
BACKWARD_LIGHTS	3	Luci di retromarcia (bianche).
ARROW_LEFT	4	Indicatore di direzione sinistro attivo.
ARROW_RIGHT	5	Indicatore di direzione destro attivo.
SPECIAL_LIGHTS	6	Pattern di luci per retromarcia speciale.
EMERGENCY_LIGHTS	7	Pattern per errore dei motori.
DEGRADED_LIGHTS	8	Pattern per segnalare modalità degradata.

Tabella 5.4: Enumerazione REAR\_LED\_TYPE

**REAR\_SIGN\_TYPE**

L'enumerazione REAR\_SIGN\_TYPE gestisce la segnaletica luminosa dell'insegna posteriore del rover.

Nome	Valore	Descrizione
SIGN_OFF	0	Segnalatore spento.
SIGN_WHITE	1	Segnalazione luminosa bianca.
SIGN_GREEN	2	Segnalazione luminosa verde.
SIGN_ORANGE	3	Segnalazione luminosa arancione.
SIGN_RED	4	Segnalazione luminosa rossa.
SIGN_YELLOW	5	Segnalazione luminosa gialla.

Tabella 5.5: Enumerazione REAR\_SIGN\_TYPE

### 5.1.3 Diagnostica e ambiente

**WORKING\_STATUS\_TYPE**

L'enumerazione WORKING\_STATUS\_TYPE rappresenta lo stato operativo decisionale della scheda di controllo.

Nome	Valore	Descrizione
NORMAL_WORKING	0	Funzionamento nominale, nessun errore rilevato.
CRITICAL_VOLTAGE_WORKING	1	Stato di protezione per sottotensione della batteria.
MOTOR_ERROR_WORKING	2	Stato di errore dovuto a guasto motori.

Tabella 5.6: Enumerazione WORKING\_STATUS\_TYPE

**MOVING\_OBSTACLE\_TYPE**

L'enumerazione MOVING\_OBSTACLE\_TYPE classifica la presenza e la direzione di eventuali ostacoli dinamici rilevati nell'area circostante il rover.

Nome	Valore	Descrizione
NO_OBSTACLE	0	Nessun ostacolo dinamico rilevato.
MOVING_FROM_LEFT	1	Ostacolo in movimento rilevato dal lato sinistro.
MOVING_FROM_RIGHT	2	Ostacolo in movimento rilevato dal lato destro.
MOVING_FROM_BOTH	3	Ostacoli in movimento rilevati da entrambi i lati.

Tabella 5.7: Enumerazione MOVING\_OBSTACLE\_TYPE

## 5.2 Bus di Stato

Le informazioni scambiate tra le due board e utilizzate dalla logica di supervisione sono organizzate mediante strutture dati aggregate definite tramite oggetti **Simulink.Bus**.

Ciascun bus rappresenta un insieme coerente di variabili eterogenee, tipizzate nel modello e utilizzate direttamente nel codice C generato automaticamente. La suddivisione dei bus riflette l'architettura del sistema, distinguendo tra stato locale delle singole board e stato globale aggregato, che costituisce la base informativa per le decisioni della supervisione.

### 5.2.1 Stato locale della Board 1

Lo **Stato locale della Board 1** rappresenta l'insieme delle informazioni acquisite direttamente dall'unità **Board 1**, dedicate principalmente al monitoraggio dello stato energetico e dinamico del rover.

Tali variabili derivano dalla sensoristica di bordo, ossia encoder dei motori, misura di tensione e temperatura, e vengono trasmesse alla **Board 2** durante la fase di *scambio dello stato locale*.

Variabile	Tipo	Descrizione
battery_voltage	single	Tensione della batteria principale del rover, utilizzata per il monitoraggio energetico e per l'attivazione degli stati di protezione in condizioni di sottotensione.
temperature	single	Temperatura interna del microcontrollore della <b>Board 1</b> , impiegata per finalità diagnostiche e per eventuali strategie di limitazione della velocità.
velocity_FA	int16	Velocità angolare del motore Front-Left, derivata dalla lettura degli encoder.
velocity_FB	int16	Velocità angolare del motore Front-Right, derivata dalla lettura degli encoder.
velocity_BA	int16	Velocità angolare del motore Back-Left, derivata dalla lettura degli encoder.
velocity_BB	int16	Velocità angolare del motore Back-Right, derivata dalla lettura degli encoder.
motorError_FA	boolean	Flag di errore associato al motore Front-Left, impostato in caso di fault rilevato dal sistema di controllo locale.
motorError_FB	boolean	Flag di errore associato al motore Front-Right.
motorError_BA	boolean	Flag di errore associato al motore Back-Left.
motorError_BB	boolean	Flag di errore associato al motore Back-Right.

Tabella 5.8: Struttura del bus **StateBusB1**

### 5.2.2 Stato locale della Board 2

Lo **Stato Locale della Board 2** raccoglie le informazioni di percezione ambientale e di comando utente. Tali variabili costituiscono la base informativa primaria per la navigazione del rover e per l'interazione con l'operatore.

Le grandezze acquisite includono misure provenienti dai sensori di bordo (giroscopio e sonar), oltre ai segnali di comando generati dal radiocomando. L'insieme di tali informazioni viene utilizzato dalla logica di supervisione per la determinazione dello stato operativo e per il calcolo delle azioni di controllo.

Variabile	Tipo	Descrizione
gyroYaw	single	Velocità angolare misurata dal giroscopio lungo l'asse verticale (asse z), utilizzata per la stima dell'orientamento dinamico del rover.
sonar1	uint16	Distanza misurata dal sensore a ultrasuoni 1 (cm), impiegata per il rilevamento di ostacoli frontali o laterali.
sonar2	uint16	Distanza misurata dal sensore a ultrasuoni 2 (cm), utilizzata per la percezione dell'ambiente circostante.
sonar3	uint16	Distanza misurata dal sensore a ultrasuoni 3 (cm), integrata nella logica di rilevamento ostacoli.
controller_y	uint16	Valore analogico dell'asse Y del joystick (acceleratore), rappresentante la richiesta di velocità longitudinale.
controller_x	uint16	Valore analogico dell'asse X del joystick (sterzo), rappresentante la richiesta di variazione direzionale.
button1	boolean	Stato logico del pulsante 1 del radiocomando.
button2	boolean	Stato logico del pulsante 2 del radiocomando.
button3	boolean	Stato logico del pulsante 3 del radiocomando.
button4	boolean	Stato logico del pulsante 4 del radiocomando.
r_stick_button	boolean	Stato logico del pulsante integrato nello stick destro.
l_stick_button	boolean	Stato logico del pulsante integrato nello stick sinistro.
controller_battery	uint8	Livello di carica della batteria del radiocomando.
controllerError	boolean	Flag di errore del sistema di controllo remoto.
gyroError	boolean	Flag di errore associato al sensore giroscopico.

Tabella 5.9: Struttura del bus **StateBusB2**

### 5.2.3 Stato globale

Lo **Stato globale** è ottenuto mediante aggregazione deterministica dello stato locale della Board 1 e dello stato locale della Board 2, integrata con variabili logiche derivate calcolate localmente dalle rispettive unità di controllo.

Tale struttura rappresenta una visione coerente e completa dello stato operativo del rover a livello di supervisione, includendo informazioni fisiche, ambientali e decisionali necessarie alla corretta evoluzione della macchina a stati.

La coincidenza bit-a-bit dello stato globale tra le due board costituisce una condizione necessaria per la prosecuzione del funzionamento cooperativo del sistema. Eventuali discrepanze tra le due rappresentazioni vengono interpretate come condizioni anomale, potenzialmente indicative di errore di comunicazione o di malfunzionamento locale.

Variabile	Tipo	Descrizione
stateB1	Bus: StateBusB1	Struttura contenente lo stato locale della Board 1.
stateB2	Bus: StateBusB2	Struttura contenente lo stato locale della Board 2.
mov_obs	MOVING_OBSTACLE_TYPE	Classificazione della presenza e direzione di eventuali ostacoli dinamici.
spc_retro	boolean	Flag logico per l'abilitazione di manovre speciali in retromarcia.
limit_vel	boolean	Flag di sicurezza che impone un limite alla velocità massima del rover.
change_vel	int8	Richiesta di incremento o decremento della velocità target.
obs_detection	boolean	Indicatore logico di rilevamento ostacoli attivo.

Tabella 5.10: Struttura del bus GSBus

### 5.2.4 Decisione

La struttura di **decisione** comprende l'insieme dei comandi calcolati dalla logica di supervisione e destinati agli attuatori del sistema.

Tale insieme di variabili costituisce l'interfaccia tra la macchina a stati di alto livello e le unità di controllo locale delle due board, traducendo lo stato globale e le condizioni operative correnti in riferimenti di velocità, modalità di funzionamento e configurazioni di segnalazione luminosa.

Nel modello Simulink questa struttura è definita mediante il bus DecBus, la cui composizione è riportata nella Tabella seguente.

Variabile	Tipo	Descrizione
rif_FA	single	Riferimento di velocità per il motore Front-Left, espresso come valore target per il controllo locale.
rif_FB	single	Riferimento di velocità per il motore Front-Right.
rif_BA	single	Riferimento di velocità per il motore Back-Left.
rif_BB	single	Riferimento di velocità per il motore Back-Right.
brk_mode	BRAKING_TYPE	Modalità di frenata da applicare ai motori.
led_A	LED_TYPE	Comando di configurazione del LED anteriore A.
led_B	LED_TYPE	Comando di configurazione del LED anteriore B.
rear_led	REAR_LED_TYPE	Configurazione della fanaleria posteriore.
rear_sign	REAR_SIGN_TYPE	Configurazione della segnaletica luminosa posteriore.
mode	ROVER_MODE	Modalità operativa corrente del rover.
relay	boolean	Segnale di attivazione del relè di potenza.
mux	boolean	Segnale di selezione del multiplexer di comando.

Tabella 5.11: Struttura del bus DecBus

### 5.3 Strutture di comunicazione

I bus definiti nelle sezioni precedenti rappresentano strutture logiche interne al modello di supervisione. Per lo scambio di dati tra le due board, tali strutture sono incapsulate in pacchetti trasmessi sul canale seriale.

Ogni pacchetto include un payload, corrispondente a uno dei bus del modello, e un codice di controllo (**CRC**) per la verifica dell'integrità dei dati.

L'aggiornamento dello stato (locale o globale) sulla board destinataria avviene esclusivamente dopo la corretta validazione del CRC.

Pacchetto	Payload	Scopo
PacketStateB1	StateBusB1 + CRC	Trasmissione dello stato locale della Board 1 verso la Board 2.
PacketStateB2	StateBusB2 + CRC	Trasmissione dello stato locale della Board 2 verso la Board 1.
PacketGstate	GState + CRC	Trasmissione dello stato globale aggregato tra le due unità.
PacketDecision	DecBus + CRC	Trasmissione dei comandi decisionali verso le unità di controllo locali.

Tabella 5.12: Strutture di comunicazione tra le due board

---

# Modellazione della supervisione

---

Nel Capitolo 5 sono state definite le strutture dati e le variabili di stato utilizzate dal sistema, mentre nel Capitolo 4 è stato progettato il protocollo di comunicazione tra le due board.

Il presente capitolo descrive la formalizzazione di tale progettazione all'interno dell'ambiente **Simulink/Stateflow**, illustrando come i diagrammi di stato ad alto livello siano stati tradotti in un modello eseguibile.

Vengono pertanto analizzate:

- l'organizzazione dei chart Stateflow relativi alle due board;
- la struttura gerarchica e parallela delle macchine a stati;
- i meccanismi di sincronizzazione, timeout e gestione delle anomalie;
- l'architettura decisionale implementata nei macro-stati funzionali.

Prima di procedere con l'analisi del modello, si riporta una breve sintesi delle scelte adottate a livello fisico, in quanto tali vincoli influenzano direttamente la struttura temporale della macchina a stati.

## 6.1 Scelta del protocollo fisico di comunicazione

L'interconnessione tra le due unità di elaborazione (Board 1 e Board 2) è realizzata mediante interfaccia **UART** (Universal Asynchronous Receiver-Transmitter), configurata con baud rate pari a 115200 bps, frame dati a 8 bit, nessuna parità, 1 bit di start e 1 bit di stop (configurazione 8N1).

Tale scelta rappresenta un compromesso tra velocità di trasferimento e robustezza ai disturbi, mantenendo contenuto il carico computazionale sulle CPU. Per rispettare i vincoli real-time e disaccoppiare il carico di comunicazione dall'elaborazione della logica di supervisione, l'interfacciamento con la periferica UART avviene mediante **DMA (Direct Memory Access)**. Il controllore DMA gestisce il trasferimento dei pacchetti direttamente tra periferica e memoria RAM, evitando la gestione del singolo byte da parte della CPU.

La comunicazione è implementata in modalità **Half-Duplex**. Poiché il protocollo di supervisione è intrinsecamente sequenziale e non prevede sovrapposizione temporale tra trasmissione e ricezione, tale configurazione consente di ottimizzare l'utilizzo delle risorse hardware ridurre il numero di linee fisiche necessarie.

Oltre al canale dati e al riferimento di massa (GND) comune, l'interfaccia elettrica prevede tre linee logiche dedicate all'arbitraggio e alla sincronizzazione:

- **SESSION**: segnale gestito dal Master che delimita l'apertura e la chiusura della finestra di comunicazione;
- **M\_TALK** (Master Talk) e **S\_TALK** (Slave Talk): linee di segnalazione utilizzate per negoziare il diritto di trasmissione e implementare il meccanismo di conferma implicita (ACK).

L'integrità dei dati è garantita mediante l'unità hardware **CRC (Cyclic Redundancy Check)** integrata nel microcontrollore STM32G4, configurata con polinomio a 32 bit. L'utilizzo di un CRC a 32 bit consente di ridurre significativamente la probabilità di

collisioni rispetto alle varianti a 8 o 16 bit, aumentando la robustezza della comunicazione in presenza di disturbi elettromagnetici. L'esecuzione del calcolo tramite periferica dedicata consente inoltre di contenere il tempo di elaborazione, preservando le risorse della CPU.

I vincoli temporali e strutturali imposti dall'architettura di comunicazione costituiscono il contesto entro cui è stata modellata la macchina a stati di supervisione in ambiente Simulink/Stateflow.

Nelle sezioni seguenti si descrive la formalizzazione del sistema mediante chart gerarchici e paralleli, che implementano le modalità operative e le logiche decisionali precedentemente progettate.

## 6.2 Board 2/ Master

Facendo riferimento alla configurazione hardware illustrata nel paragrafo 1.2, il sistema elabora i segnali provenienti dalla sensoristica di bordo, che comprende i moduli di rilevamento ostacoli (tre sonar), l'accelerometro e l'interfaccia di comando remoto (controller). A livello di implementazione nell'automa, questi dati grezzi vengono pre-elaborati e mappati in un **vettore di ingresso** strutturato, essenziale per la costruzione dello Stato Locale (la cui definizione formale è riportata nel paragrafo 5.2.2). Analogamente, l'**output decisionale** prodotto dalla logica di controllo rispetta la codifica specificata nel paragrafo 5.2.4, garantendo la corretta attivazione degli attuatori.

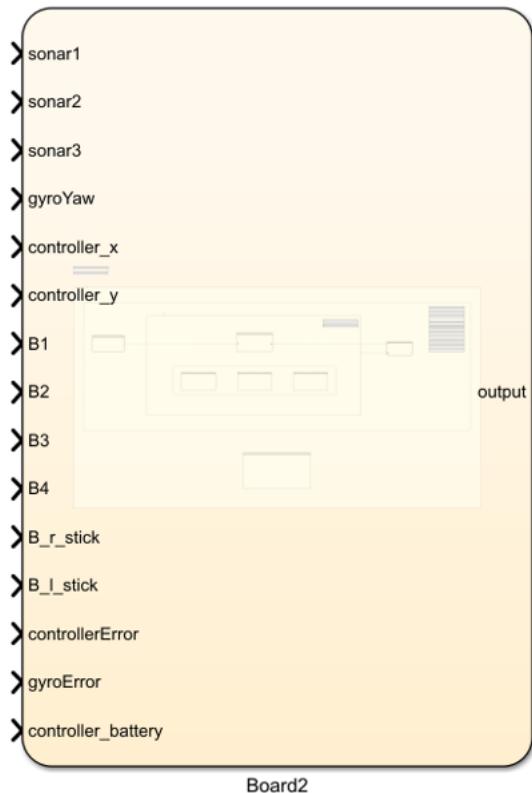


Figura 6.1: Blocco Simulink Board2

### 6.2.1 Variabili del modello

Il chart Stateflow relativo alla Board 2 (Master) utilizza un insieme strutturato di variabili organizzate secondo lo scope previsto dall'ambiente Simulink/Stateflow.

Le variabili sono suddivise nelle seguenti categorie:

- **Input**: segnali provenienti dall'esterno del chart (sensoristica locale e dati ricevuti);
- **Local**: variabili interne utilizzate per la gestione della logica e degli stati intermedi;
- **Output**: segnali prodotti dal chart verso l'esterno;
- **Constant**: parametri statici di configurazione.

Per ciascuna variabile sono riportati nome, visibilità (scope), tipo di dato e valore iniziale.

#### Input

Le variabili di **Input** rappresentano i segnali in ingresso al chart, provenienti dall'ambiente esterno, da sensori fisici o da dispositivi di controllo. Esse costituiscono le informazioni di base su cui il sistema prende le proprie decisioni operative.

Nome	Visibilità	Tipo	Valore Iniziale
sonar1	Input	uint16	—
sonar2	Input	uint16	—
sonar3	Input	uint16	—
controller_x	Input	uint16	—
controller_y	Input	uint16	—
gyroYaw	Input	single	—
controller_battery	Input	uint8	—
controllerError	Input	boolean	—
gyroError	Input	boolean	—
B1	Input	boolean	—
B2	Input	boolean	—
B3	Input	boolean	—
B4	Input	boolean	—
B_l_stick	Input	boolean	—
B_r_stick	Input	boolean	—

Tabella 6.1: Variabili di Input del chart Stateflow – Board 2

#### Local

Le variabili **Local** sono utilizzate internamente al chart per memorizzare stati intermedi, informazioni temporanee e risultati parziali della logica di controllo. Esse non sono visibili all'esterno del modello e servono a supportare il funzionamento della macchina a stati.

Nome	Visibilità	Tipo	Valore Iniziale
retransmitted	Local	uint8	–
state	Local	Bus: StateBusB2	–
global_state	Local	Bus: GSBus	–
decision	Local	Bus: DecBus	–
special_retro	Local	boolean	–
angle	Local	single	–
limit_velocity	Local	boolean	–
receivedStatePacket	Local	Bus: PacketStateB1	–
receivedDecisionPacket	Local	Bus: PacketDecision	–
receivedGlobalStatePacket	Local	Bus: PacketGstate	–
receivedPing	Local	uint8	–
prevYaw	Local	single	–
time_obs_s1	Local	uint32	–
time_obs_s3	Local	uint32	–
time_temp	Local	uint32	–
time_comm	Local	uint32	–
time_button	Local	uint32	–
obs_detection	Local	boolean	–
distance_threshold	Local	uint16	–
max_velocity	Local	uint8	–
change_velocity	Local	int8	–
moving_from_left	Local	boolean	–
moving_from_right	Local	boolean	–
special_retro_rotating	Local	boolean	–
turn_counter	Local	uint8	–
working_status	Local	Enum: WORKING_STATUS_TYPE	–
open_loop	Local	boolean	–
panic_lockdown	Local	boolean	–
init_count	Local	uint8	–
TURN_THRESHOLD	Local	single	–

Tabella 6.2: Variabili Local del chart Stateflow – Board 2

### Output

Le variabili di **Output** rappresentano i segnali prodotti dal chart verso l'esterno del modello. Nel caso in esame, l'uscita output, raccoglie le decisioni finali elaborate dal sistema e destinate ai moduli di attuazione.

Nome	Visibilità	Tipo	Valore Iniziale
output	Output	Bus: DecBus	–

Tabella 6.3: Variabile di Output del chart Stateflow – Board 2

**Constant**

Le variabili **Constant** definiscono parametri statici del sistema, utilizzati per configurare soglie, limiti operativi e temporizzazioni. Tali valori non vengono modificati durante l'esecuzione del modello e garantiscono un comportamento deterministico e riproducibile.

Nome	Visibilità	Tipo	Valore
MAX_RPM	Constant	uint8	150
LIMITED_RPM	Constant	uint8	80
MIN_RPM	Constant	uint8	50
TURN_RPM	Constant	single	20.0
TURN_BACK_RPM	Constant	single	40.0
VEL_CHANGE	Constant	uint8	10
TURN_RATIO	Constant	single	0.35
PURE_TURN_EPS	Constant	single	0.01
CENTER	Constant	single	255.0
TURN_ANGLE	Constant	single	45.0
TURN_BACK_ANGLE	Constant	single	180.0
TURN_COUNT	Constant	uint8	20
TURN_BACK_COUNT	Constant	uint32	–
TURN_LIMIT_COUNT	Constant	uint8	30
TURN_BACK_LIMIT_COUNT	Constant	uint8	50
MIN_TURN_SCALE_EVASIVE	Constant	single	0.2
MAX_TURN_SCALE_EVASIVE	Constant	single	–

Tabella 6.4: Parametri cinematici e dinamici – Board 2

Nome	Visibilità	Tipo	Valore
LOW_VOLTAGE	Constant	single	9.83
CRITICAL_VOLTAGE	Constant	single	9.0
HIGH_TEMPERATURE	Constant	single	100.0
LOW_CONTROLLER_BATTERY	Constant	uint8	5
MIN_DISTANCE	Constant	uint16	150
MAX_DISTANCE	Constant	uint16	300
IMM_DISTANCE	Constant	uint16	70
PROTECTION_DISTANCE	Constant	uint16	40
STOP_THRESHOLD	Constant	uint16	1
INIT_COUNTER	Constant	uint8	35
PERIOD	Constant	single	0.06

Tabella 6.5: Parametri energetici e soglie di sicurezza – Board 2

Nome	Visibilità	Tipo	Valore
OBS_TIMEOUT	Constant	uint32	1000
TEMP_TIMEOUT	Constant	uint32	10000
BUTTON_TIMEOUT	Constant	uint32	700
BYTE_SEND_TIMEOUT	Constant	uint32	500
BYTE_RECEIVE_TIMEOUT	Constant	uint32	1000
DECISION_SEND_TIMEOUT	Constant	uint32	3000
DECISION_RECEIVE_TIMEOUT	Constant	uint32	3500
STATE_SEND_TIMEOUT	Constant	uint32	3000
STATE_RECEIVE_TIMEOUT	Constant	uint32	3100
GLOBAL_STATE_SEND_TIMEOUT	Constant	uint32	5400
GLOBAL_STATE_RECEIVE_TIMEOUT	Constant	uint32	5900
INITIAL_TIMEOUT	Constant	uint32	1500
WAIT_TIMEOUT	Constant	uint32	500
MAX_RETRANSMIT	Constant	uint8	1

Tabella 6.6: Parametri temporali e di protocollo – Board 2

### 6.2.2 Struttura architetturale

Lo schema è strutturato secondo una **decomposizione parallela (AND states)**. Al livello gerarchico più alto, il diagramma si divide in due macro-stati: **Board\_state** e **Board\_decision**. Questa separazione architettonica permette di isolare la **logica di stato** (protocollo e salute del sistema) dalla **logica di attuazione**.

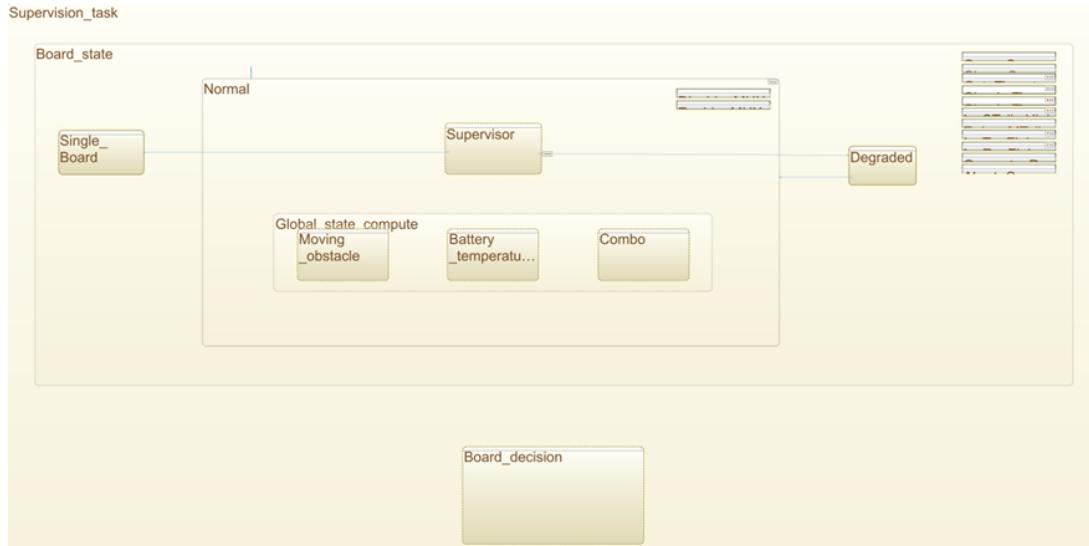


Figura 6.2: Supervision\_task in Simulink Board2

#### **Board\_decision**

Il blocco **Board\_decision** ha il compito di sintetizzare la decisione finale di attuazione, elaborando le informazioni derivanti dallo stato globale corrente. Grazie alla decomposizione parallela rispetto alla macchina a stati operativa (**Board\_state**), questo modulo è indipendente dai transitori di comunicazione. Il sistema, quindi, mantiene attive le ultime variabili di controllo valide, preservando la "memoria" delle azioni intraprese immediatamente prima di eventuali interruzioni. Si rimanda alla sezione 6.4.4 per dettagli sul funzionamento nello specifico.

#### **Board\_state**

Il blocco **Board\_state** ha il compito di tracciare la condizione operativa corrente della scheda in relazione alla comunicazione con la controparte. Al suo interno, la logica è sequenziale (**OR decomposition**) e si articola nelle seguenti modalità:

- **Normal:** è lo stato complesso di cooperazione nominale. Contiene al suo interno un'ulteriore gerarchia:
  - **Supervisor:** gestisce il protocollo di comunicazione (**handshake**, scambio dati).
  - **Global\_state\_compute:** elabora i dati dei sensori per aggiornare lo stato globale della board.
- **Degraded:** gestisce la perdita temporanea di comunicazione e i tentativi di ripristino.
- **Single\_Board:** gestisce la condizione di isolamento permanente in caso di guasto della controparte.

**Normal** Lo stato **Normal** è implementato come uno stato composito a **decomposizione parallela (AND-state)**, all'interno del quale vengono eseguiti in concorrenza quattro automi distinti, ciascuno preposto a specifiche funzioni logiche:

1. **Supervisor**: è l'orchestratore principale, responsabile della sincronizzazione del protocollo di comunicazione e della gestione delle transizioni di stato del sistema.
2. **Global\_state\_compute**: contiene i sottosistemi di calcolo (**Moving\_obstacle**, **Battery\_temperature\_manager**, **Combo**) dedicati all'elaborazione dei dati sensoriali grezzi per la costruzione delle variabili che compongono lo **Stato Globale**.

All'ingresso nel super-stato (**Entry Action**), la logica impone il segnale di controllo del Multiplexer (**MUX**) hardware per **delegare il controllo dei motori allo Slave**. Viceversa, in fase di uscita (**Exit Action**) il sistema resetta il MUX, revocando la delega e riassumendo il controllo diretto dei motori in modalità **Open Loop**, al fine di garantire la sicurezza della manovra anche in assenza di cooperazione.

**Supervisor** L'automa **Supervisor** costituisce il **core deterministico** del sistema, gestendo l'intera sequenza del protocollo di cooperazione **Master–Slave**. La sua struttura a stati finiti scandisce rigorosamente le fasi di trasmissione, attesa e validazione dei messaggi, governando le transizioni operative sulla base degli esiti dei controlli di integrità (**CRC**) e della gestione delle conferme, in linea con quanto descritto in fase progettuale. Un aspetto critico per la robustezza del sistema è la gestione dei tempi di risposta. Ad ogni stato del **Supervisor** è associato un **timer di timeout** dedicato:

- **All'ingresso dello stato**: viene inizializzato il contatore temporale.
- **Durante l'esecuzione**: il sistema verifica costantemente il rispetto della soglia prevista.
- **In caso di violazione**: qualora l'evento atteso non si verifichi entro il tempo limite, l'automa forza una transizione di gestione dell'errore. Questa innesta un tentativo di ritrasmissione o, in alternativa, il passaggio allo stato **Degraded**, garantendo che il sistema non rimanga mai bloccato in attese indefinite (**deadlock**).

Il ciclo di esecuzione del **Supervisor** coordina gli altri automi paralleli tramite un meccanismo a eventi:

1. **Costruzione Stato Globale**: una volta avvenuta la ricezione validata dello stato locale dalla controparte, il Supervisor emette l'evento **STEP**. Questo trigger attiva l'esecuzione sincrona delle macchine **Moving\_obstacle**, **Combo** e **Battery\_temperature\_manager** per la sintesi dello **Stato Globale** aggiornato.
2. **Calcolo Decisione**: completato con successo lo scambio e il confronto dello Stato Globale, viene generato un nuovo evento **STEP** indirizzato all'automa **Board\_decision**, che procede al calcolo della decisione di attuazione.

Se anche la fase di negoziazione della decisione si conclude positivamente (consenso raggiunto), la transizione termina e l'automa si predisponde all'esecuzione del ciclo successivo.

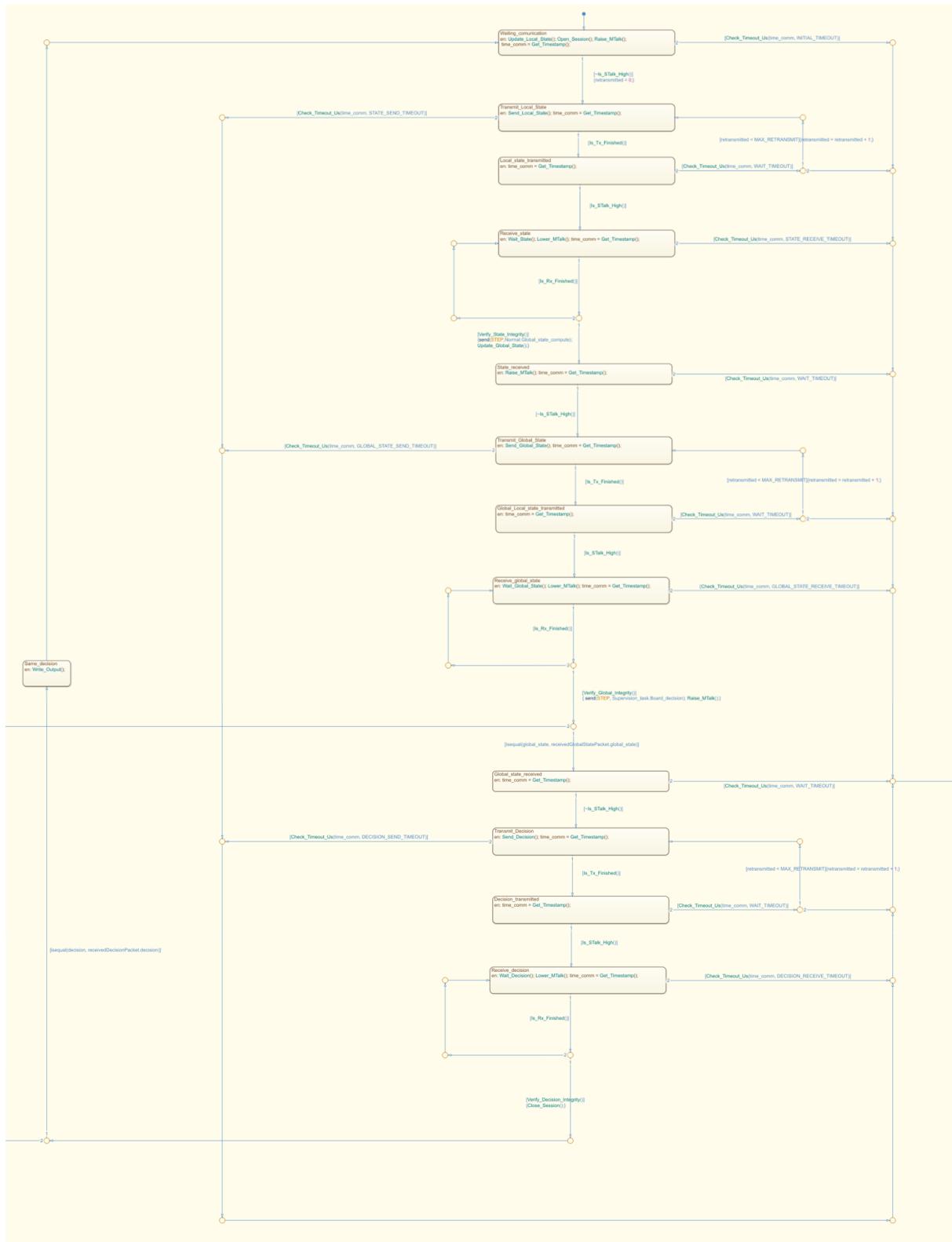


Figura 6.3: Supervisor Board 2

**Degraded** All'attivazione dello stato **Degraded**, l'automa interrompe il flusso di dati verso l'esterno e riconfigura la logica di controllo commutando su una **strategia di fallback locale**. Inibita la cooperazione distribuita, l'algoritmo di attuazione elabora la decisione basandosi esclusivamente sulle **variabili di stato interne** e sulle **risorse hardware** direttamente accessibili alla singola board.

Il calcolo della decisione segue un rigoroso approccio gerarchico basato sulla **priorità dei rischi (Priority-Based Control Logic)**. La catena di controllo valuta sequenzialmente una serie di condizioni di guardia, ordinate dalla più critica alla meno restrittiva, secondo il seguente schema:

- **Diagnostica Critica e Kill Switch (Priorità 1 – Massima)**: viene preliminarmente verificato lo stato di salute dell'hardware. Qualora i driver motore segnalino un guasto interno o l'operatore prema la combinazione di panico (**B1+B2+B3+B4** simultaneamente), il sistema forza lo **Shutdown Totale**. L'azione consiste nell'azzeramento dei riferimenti, nell'attivazione della frenata di emergenza e, soprattutto, nell'**apertura fisica del relè di potenza**.
- **Sicurezza Anti-Collisione (Priorità 2)**: in assenza di guasti critici, si analizzano i sensori di prossimità con una logica condizionale alla manovra. Se un sensore rileva un ostacolo sotto la soglia di sicurezza e contemporaneamente l'utente sta comandando un avanzamento, il sistema impone l'**arresto immediato**.
- **Arresto di Emergenza Manuale (Priorità 3)**: successivamente, si verifica lo stato del pulsante di stop dedicato. Se premuto dall'operatore, viene replicata la procedura di **soppressione istantanea del moto** con attivazione della frenata di emergenza.
- **Arresto Operativo e Monitoraggio Batteria (Priorità 4)**: si valuta la necessità di una pausa controllata. Tale condizione si verifica se viene premuto il tasto di arresto operativo o qualora la batteria del controller sia scesa sotto la soglia critica. In questo scenario, il sistema comanda una **frenata smooth** con azzeramento del riferimento di velocità.
- **Guida Manuale in Open-Loop (Priorità 5 – Minima)**: solo qualora nessuna delle condizioni di inibizione precedenti sia soddisfatta, il sistema abilita la movimentazione. I comandi del joystick vengono elaborati per calcolare i riferimenti dei singoli motori, permettendo la navigazione in assenza di comunicazione.

Nel ciclo di esecuzione successivo, la board reitererà questa procedura decisionale autonoma, tentando successivamente, come descritto nella sezione relativa al protocollo, di **ripristinare il canale di comunicazione** con l'altra board.

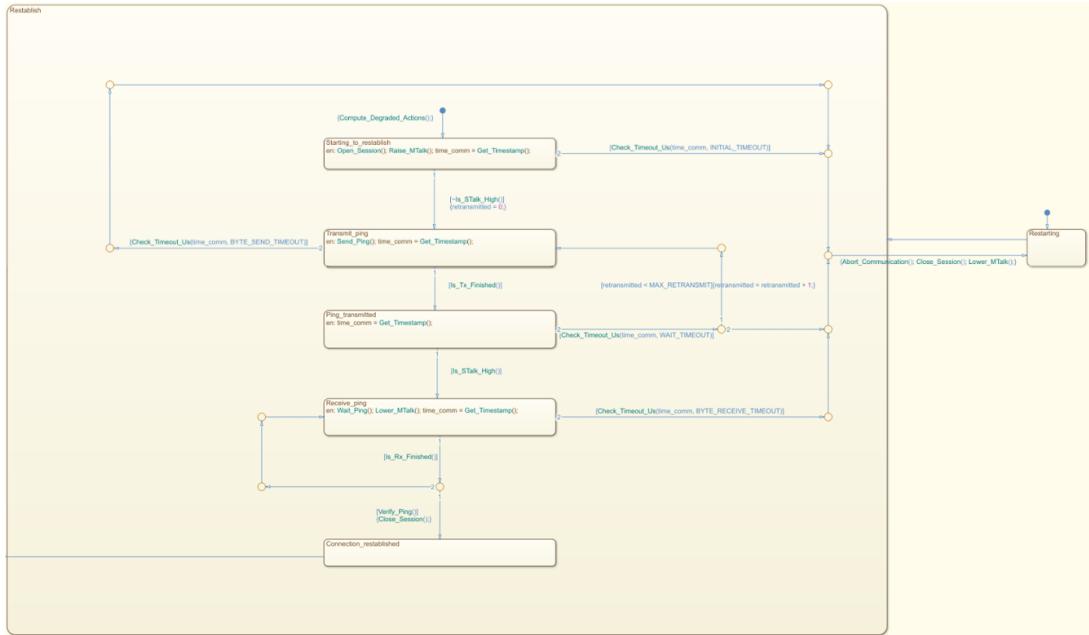


Figura 6.4: Restablish Board 2

**Single\_board** In questa modalità operativa, la board replica funzionalmente la medesima **logica decisionale** descritta per lo stato **Degraded**, applicando la stessa **gerarchia di priorità** esclusivamente sulle letture locali, al fine di garantire la conduzione del veicolo in sicurezza anche in isolamento.

La differenza sostanziale risiede nella gestione del **flusso di esecuzione post-decisionale**. A differenza della modalità **Degraded**, che tenta ciclicamente il ripristino del canale, in **Single-Board** il sistema assume l'assoluta **inaffidabilità della controparte**. Di conseguenza, una volta attuata la decisione sui motori, il ciclo di controllo termina immediatamente senza innescare alcuna procedura di comunicazione. L'unità prosegue quindi in un **loop di controllo puramente locale**, senza attendere né inviare alcun messaggio verso l'esterno.

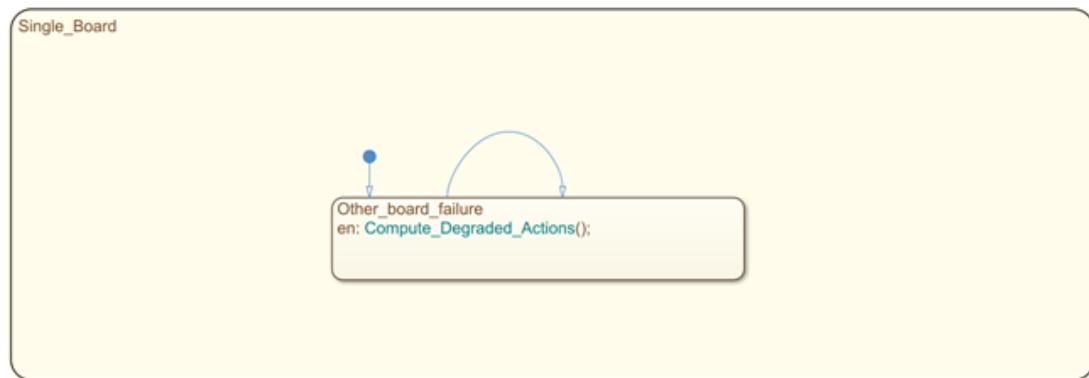


Figura 6.5: SingleBoard Board2

## 6.3 Board 1/ Slave

Facendo riferimento alla configurazione hardware illustrata nel paragrafo 1.2, il sistema elabora i segnali provenienti dalla sensoristica di bordo, che comprende il sensore di temperatura interno, il modulo di monitoraggio della batteria e i 4 encoder dei motori. A livello di implementazione nell'automa, questi dati grezzi vengono pre-elaborati e mappati in un vettore di ingresso strutturato, essenziale per la costruzione dello **Stato Locale** (la cui definizione formale è riportata nel paragrafo 5.2.1). Analogamente, l'**output decisionale** prodotto dalla logica di controllo rispetta la codifica specificata nel paragrafo 5.2.4, garantendo la corretta attivazione degli attuatori.



Figura 6.6: Blocco Simulink Board1

### 6.3.1 Variabili del Modello

Le seguenti tabelle riportano l'elenco completo delle variabili utilizzate nel chart Stateflow relativo alla **Board1 (Slave)**, organizzate in base allo scope definito nel modello Simulink/Stateflow. Per ciascuna variabile sono specificati il nome, la visibilità, il tipo di dato e l'eventuale valore iniziale.

## Input

Le variabili di **Input** rappresentano i segnali in ingresso al chart, provenienti dall'ambiente esterno e dalla sensoristica locale. Esse costituiscono le informazioni primarie su cui la Board Slave costruisce il proprio Stato Locale e valuta le condizioni di sicurezza.

Nome	Visibilità	Tipo	Valore Iniziale
temperature	Input	single	–
battery_voltage	Input	single	–
velocity_BA	Input	int16	–
velocity_BB	Input	int16	–
velocity_FA	Input	int16	–
velocity_FB	Input	int16	–
motorError_FA	Input	boolean	–
motorError_FB	Input	boolean	–
motorError_BA	Input	boolean	–
motorError_BB	Input	boolean	–

Tabella 6.7: Variabili di Input del chart Stateflow – Board 1

## Local

Le variabili **Local** sono utilizzate internamente al chart per memorizzare stati intermedi, informazioni temporanee e risultati parziali della logica di controllo. Non sono visibili all'esterno del modello e supportano il corretto funzionamento della macchina a stati.

Tabella 6.8: Variabili Local del chart Stateflow – Board 1

Nome	Visibilità	Tipo	Valore Iniziale
retransmitted	Local	uint8	–
state	Local	Bus: StateBusB1	–
global_state	Local	Bus: GSBus	–
decision	Local	Bus: DecBus	–
special_retro	Local	boolean	–
angle	Local	single	–
limit_velocity	Local	boolean	–
receivedDecisionPacket	Local	Bus: PacketDecision	–
receivedGlobalStatePacket	Local	Bus: PacketGstate	–
receivedStatePacket	Local	Bus: PacketStateB2	–
receivedPing	Local	uint8	–
prevYaw	Local	single	–
time_obs_s1	Local	uint32	–
time_obs_s3	Local	uint32	–
time_temp	Local	uint32	–

Nome	Visibilità	Tipo	Valore Iniziale
time_comm	Local	uint32	–
time_button	Local	uint32	–
obs_detection	Local	boolean	–
distance_threshold	Local	uint16	–
MIN_RPM	Local	uint8	–
max_velocity	Local	uint8	–
change_velocity	Local	int8	–
moving_from_left	Local	boolean	–
moving_from_right	Local	boolean	–
special_retro_rotating	Local	boolean	–
turn_counter	Local	uint8	–
working_status	Local	Enum: WORKING_STATUS_TYPE	–
open_loop	Local	boolean	–
init_count	Local	uint8	–

### Output

La variabile di **Output** rappresenta il segnale prodotto dal chart verso l'esterno del modello. Nel caso in esame, l'uscita `output` raccoglie le decisioni finali elaborate dal sistema e destinate ai moduli di attuazione.

Nome	Visibilità	Tipo	Valore Iniziale
output	Output	Bus: DecBus	–

Tabella 6.9: Variabili di Output del chart Stateflow – Board 1

**Constant**

Le variabili **Constant** definiscono parametri statici del sistema, utilizzati per configurare soglie, limiti operativi e temporizzazioni. Tali valori non vengono modificati durante l'esecuzione del modello e garantiscono un comportamento deterministico e riproducibile.

Nome	Visibilità	Tipo	Valore
MAX_RPM	Constant	uint8	150
LIMITED_RPM	Constant	uint8	80
TURN_RPM	Constant	single	20.0
TURN_BACK_RPM	Constant	single	40.0
VEL_CHANGE	Constant	uint8	10
TURN_RATIO	Constant	single	0.35
PURE_TURN_EPS	Constant	single	0.01
CENTER	Constant	single	255.0
TURN_ANGLE	Constant	single	45.0
TURN_BACK_ANGLE	Constant	single	180.0
TURN_COUNT	Constant	uint8	20
TURN_BACK_COUNT	Constant	uint32	–
TURN_LIMIT_COUNT	Constant	uint8	30
TURN_BACK_LIMIT_COUNT	Constant	uint8	50
MIN_TURN_SCALE_EVASIVE	Constant	single	0.2
MAX_TURN_SCALE_EVASIVE	Constant	single	–

Tabella 6.10: Parametri cinematici e dinamici – Board 1

Nome	Visibilità	Tipo	Valore
LOW_VOLTAGE	Constant	single	9.83
CRITICAL_VOLTAGE	Constant	single	9.0
HIGH_TEMPERATURE	Constant	single	100.0
LOW_CONTROLLER_BATTERY	Constant	uint8	5
MIN_DISTANCE	Constant	uint16	150
MAX_DISTANCE	Constant	uint16	300
IMM_DISTANCE	Constant	uint16	70
PROTECTION_DISTANCE	Constant	uint16	40
STOP_THRESHOLD	Constant	uint16	1
INIT_COUNTER	Constant	uint8	35
PERIOD	Constant	single	0.06

Tabella 6.11: Parametri energetici e soglie di sicurezza – Board 1

Nome	Visibilità	Tipo	Valore
OBS_TIMEOUT	Constant	uint32	1000
TEMP_TIMEOUT	Constant	uint32	10000
BUTTON_TIMEOUT	Constant	uint32	700
BYTE_SEND_TIMEOUT	Constant	uint32	500
BYTE_RECEIVE_TIMEOUT	Constant	uint32	1000
DECISION_SEND_TIMEOUT	Constant	uint32	3000
DECISION_RECEIVE_TIMEOUT	Constant	uint32	3500
STATE_SEND_TIMEOUT	Constant	uint32	2600
STATE_RECEIVE_TIMEOUT	Constant	uint32	3500
GLOBAL_STATE_SEND_TIMEOUT	Constant	uint32	5400
GLOBAL_STATE_RECEIVE_TIMEOUT	Constant	uint32	5900
WAIT_TIMEOUT	Constant	uint32	500
MAX_RETRANSMIT	Constant	uint8	1

Tabella 6.12: Parametri temporali e di protocollo – Board 1

### 6.3.2 Struttura architetturale

Coerentemente con l'architettura definita per l'unità **Master**, il diagramma della **Board Slave** replica la medesima **decomposizione parallela**. La logica si articola su due **macro-stati concorrenti**: **Board\_state**, che governa le transizioni tra le modalità operative (**Normal**, **Degraded**, **Single\_Board**), e **Board\_decision**, deputato alla **sintesi dei comandi di attuazione**.

Le definizioni funzionali e le logiche di transizione rimangono **invariate** rispetto alla controparte. Questa scelta di mantenere una **perfetta specularità** tra i diagrammi a stati garantisce che, a fronte degli stessi eventi di sistema, entrambe le unità reagiscano in modo **coerente e predicibile**, allineandosi sullo stesso **stato operativo**.

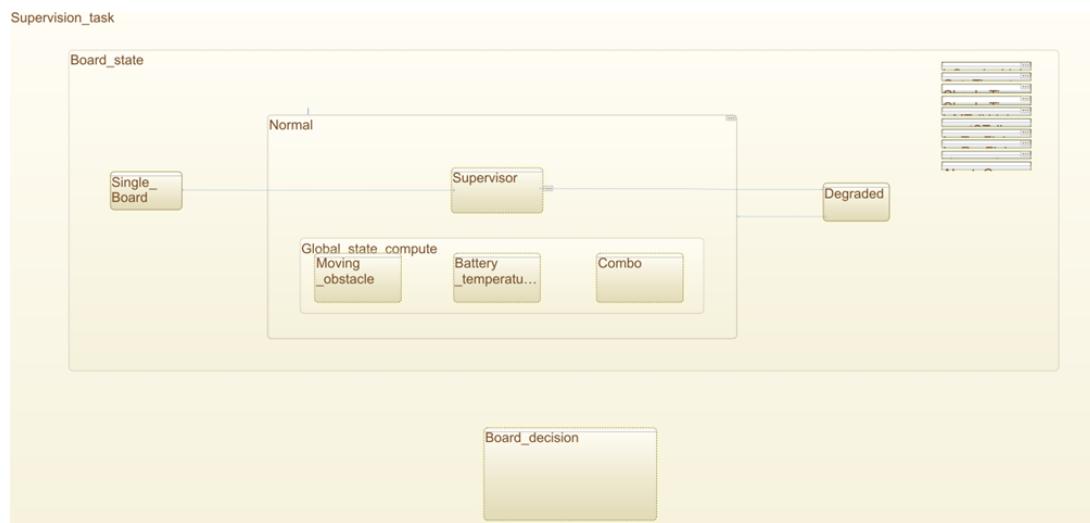


Figura 6.7: Supervision\_task in Simulink Board1

Si procede ora all’analisi dettagliata dell’architettura interna dei macro-stati, esaminandone la **decomposizione gerarchica** in sotto-automi e le logiche di controllo che ne governano l’evoluzione dinamica; poiché la definizione della logica è analoga per la board Master e per la board Slave relativamente al macro-stato **Board\_decision**, si analizzerà nel dettaglio il macro-stato **Board\_state**, rimandando al Paragrafo 6.4.4 per la descrizione di **Board\_decision**.

### **Board\_state**

**Normal** L’architettura software dello stato della **board Slave** è speculare a quella definita per l’unità **Master**. Il macro-stato **Normal** mantiene la medesima **decomposizione parallela (AND-state)**, eseguendo in concorrenza l’automa **Supervisor** e i sottosistemi di calcolo (**Moving\_obstacle**, **Combo**, **Battery\_temperature\_manager**).

**Supervisor** Anche in questo caso, il **Supervisor** agisce come **orchestratore del protocollo**, condividendo con la controparte le stesse **logiche di progettazione**. Vengono riutilizzati i medesimi **meccanismi di controllo temporale** (watchdog timer per ogni stato) per garantire la **robustezza della comunicazione** e rilevare tempestivamente condizioni di errore. Anche la **sequenza operativa** ricalca fedelmente il ciclo già analizzato:

1. Al completamento della **ricezione dello Stato Globale** della controparte, viene generato l’evento **STEP** per attivare l’aggiornamento sincrono delle macchine a stati di calcolo dello **Stato Globale**.
2. Successivamente, validata la **decisione ricevuta** dalla board **Master**, un secondo evento **STEP** innesca l’automa **Board\_decision** per l’elaborazione della **decisione di controllo**.
3. Il ciclo si conclude con la **verifica del consenso** sulla decisione finale, predisponendo la **Slave** al ciclo successivo.

Figura 6.8: Supervisor Board 1

**Degraded** All’ingresso nello stato **Degraded**, la **Board Slave** sospende le attività di comunicazione e isola la propria **logica decisionale**. A differenza dell’unità **Master**, tuttavia, lo **Slave** si trova in una condizione di **cecità sensoriale** rispetto alla navigazione: non avendo accesso diretto ai sensori di rilevamento ostacoli (**Sonar**) né all’interfaccia di comando remoto (**Controller**), l’unità non possiede le informazioni minime necessarie per garantire un movimento sicuro. Di conseguenza, il sistema adotta una politica di **Fail-Safe rigoroso**:

- **Attuazione Motori:** viene imposta una **frenata di emergenza** e i riferimenti di velocità sono **forzati a zero**, inibendo qualsiasi moto del veicolo.
  - **Segnalazione Visiva Gerarchica:** il sottosistema di illuminazione non si limita a indicare lo stato di avaria, ma segue una **logica a priorità decrescente** per determinare il feedback più appropriato:
    1. **Priorità 1 – Allerta Critica:** se la causa dell'arresto è un guasto interno ai driver, il sistema forza una **segnalazione di massima allerta** (luci rosse lampeggianti).
    2. **Priorità 2 – Protezione Energetica:** in assenza di errori motore, si verifica la **tensione di alimentazione**. Se questa scende sotto la soglia critica, il sistema spegne tutte le **segnalazioni luminose** per prevenire danni irreversibili alle celle della batteria.
    3. **Priorità 3 – Segnalazione di Stato:** se le condizioni precedenti non sussistono, il sistema comunica visivamente l'ingresso in modalità **Degraded**, impostando le luci su una **segnalazione di avvertimento gialla**.

Nei cicli operativi successivi, il sistema reitererà l'esecuzione di questa **procedura di sicurezza autonoma** (mantenimento dello stato di arresto). Contestualmente, la board continuerà a tentare il **ripristino del canale di comunicazione** con l'unità Master. Come dettagliato nella sezione relativa al protocollo, non appena il tentativo di **handshake** avrà successo e il collegamento sarà ristabilito, il sistema abbandonerà la modalità degradata per tornare al **funzionamento nominale**.

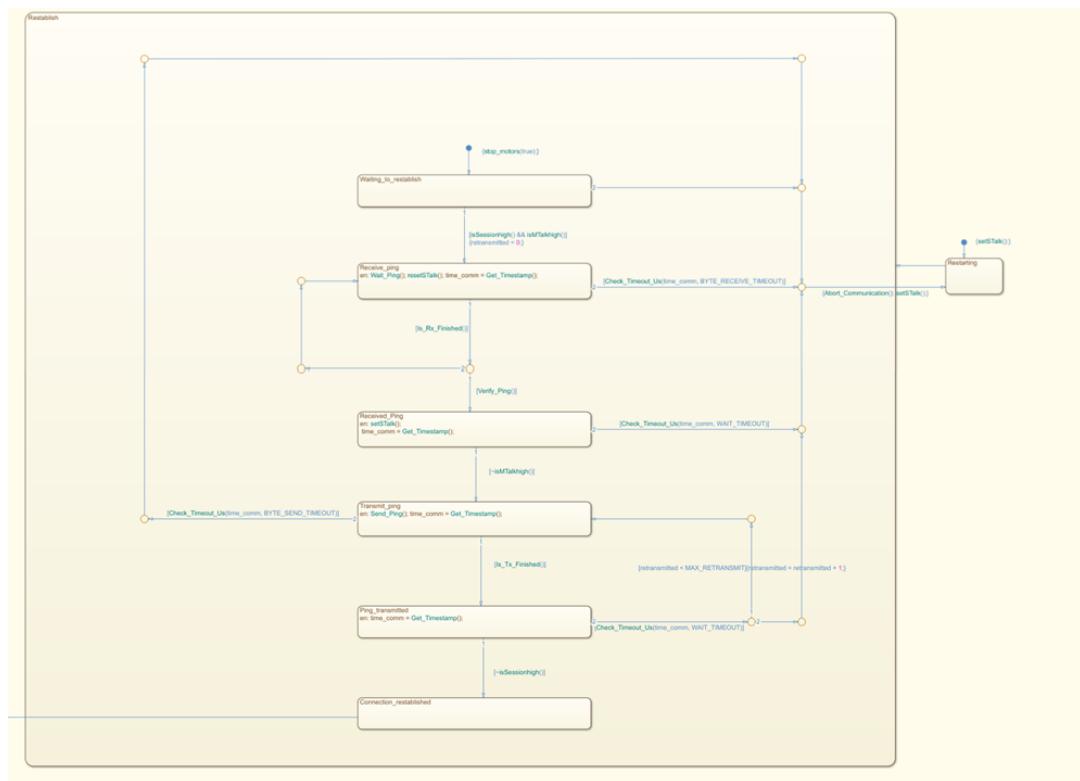


Figura 6.9: Restablish Board 1

**Single\_board** In questa configurazione, la board replica funzionalmente la **strategia di sicurezza** già adottata nello stato **Degraded**, riutilizzando la medesima **procedura di arresto meccanico** e la stessa **gerarchia di protezione** per i guasti critici. La divergenza operativa emerge esclusivamente in assenza di anomalie hardware: il sistema, infatti, non attiva la **segnalazione gialla di avvertimento**, bensì impone una **segnalazione rossa fissa**, comunicando visivamente lo stato di **isolamento permanente** rispetto alla condizione transitoria di recupero.

A livello logico, la distinzione cruciale risiede nella gestione del **ciclo di vita del sistema**: assumendo un **guasto irreversibile** dell'unità **Master**, la **Slave** non intraprende alcun tentativo di **ripristino della comunicazione**. Una volta applicate le misure di sicurezza locali, il ciclo di controllo termina immediatamente, isolando definitivamente la board e mantenendo lo **stato di arresto** in attesa di un **intervento esterno** (reset manuale).

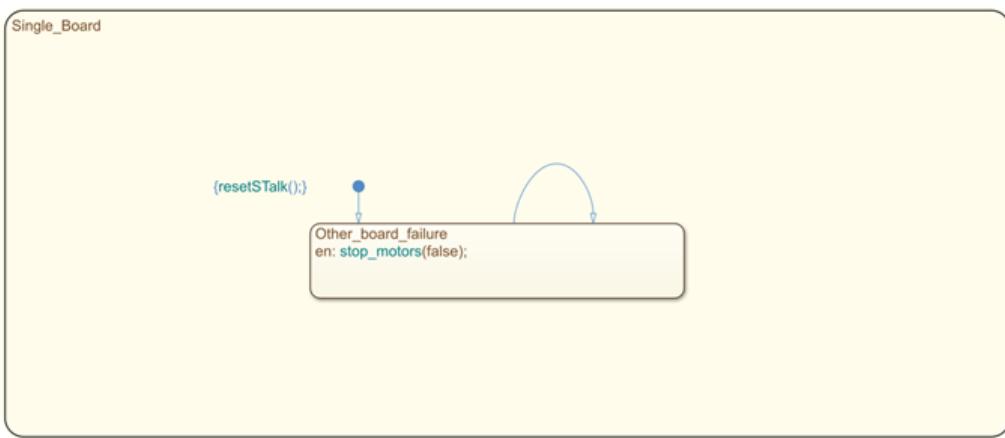


Figura 6.10: SingleBoard Board1

## 6.4 Automi comuni

I moduli descritti di seguito rappresentano **componenti logici condivisi**, istanziati identicamente sia nel firmware della **Board Master** che in quello della **Slave**. La caratteristica fondamentale di questi blocchi è il **disaccoppiamento tra la logica di controllo e i dati elaborati**: sebbene la topologia dell'automa e le regole di transizione rimangano invarianti, i vettori di ingresso su cui vengono valutate le condizioni differiscono. Grazie a questo approccio, il medesimo algoritmo può essere configurato dinamicamente per analizzare, a seconda del contesto operativo e del ruolo della board, sia le variabili dello **Stato Locale** (generate dalla sensoristica di bordo) sia quelle dello **Stato Ricevuto** (trasmesse dalla controparte), semplificando il **riuso del codice**.

Un ulteriore aspetto architetturale riguarda la **politica di esecuzione**: questi automi non operano in modalità continua, ma avanzano in modo **discreto e sincrono**. L'evoluzione dei loro stati è rigorosamente controllata dal **Supervisor**, che agisce come orchestratore invocando esplicitamente l'evento **STEP**. Ciò garantisce che l'elaborazione logica avvenga esclusivamente all'interno della **finestra temporale prevista**, assicurando il **determinismo sequenziale** dell'intero ciclo di controllo.

### 6.4.1 Moving\_obstacle

Questo automa ha il compito di analizzare la **correlazione delle letture dei sonar** su più cicli di esecuzione, al fine di discriminare **ostacoli dinamici in movimento**.

Il macro-stato è configurato a **decomposizione parallela**, eseguendo tre sotto-automi concorrenti secondo una rigorosa **gerarchia di priorità in scrittura**:

1. **No\_obstacle (Default – Priorità Minima)**: è il primo automa nell'ordine di esecuzione. La sua funzione è di **reset**: ad ogni **STEP** inizializza le variabili di output sullo stato **Nessun movimento**. Avendo la priorità di scrittura più bassa, il suo output è inteso come **valore di default**, valido solo se nessuno degli altri automi rileva eventi significativi.

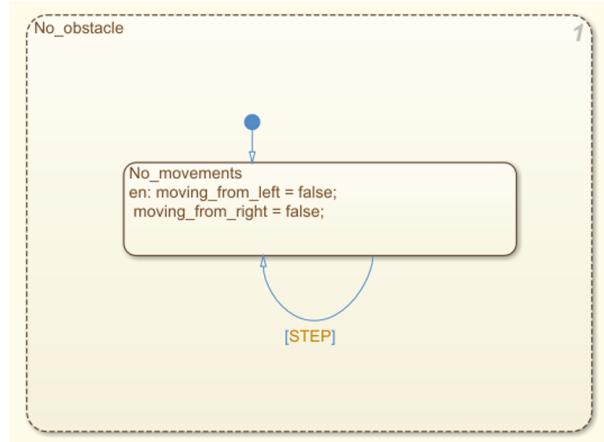


Figura 6.11: No\_obstacle

2. **Obstacle\_from\_left (Rilevamento Sinistro)**: si occupa di identificare un moto con direzione **Sinistra → Centro**. La macchina a stati valuta la sequenza temporale: se viene rilevato un ostacolo sul sensore laterale sinistro e, successivamente (entro una finestra temporale definita da **OBS\_TIMEOUT**), si attiva anche il sensore centrale, il sistema classifica l'evento come **Ostacolo in movimento da sinistra**, sovrascrivendo lo stato di default.

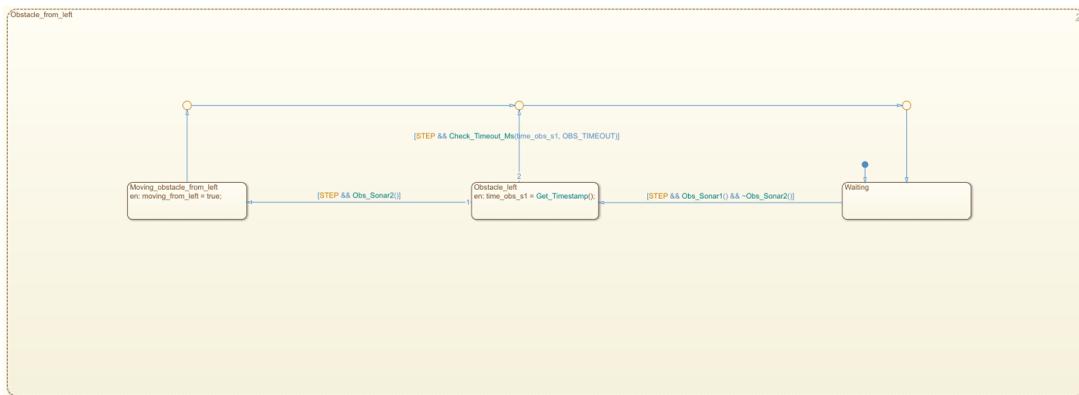


Figura 6.12: Obstacle\_from\_left

3. **Obstacle\_from\_right (Rilevamento Destro – Priorità Massima)**: il funzionamento è **speculare** al precedente, ma con sequenza **Destra → Centro**.

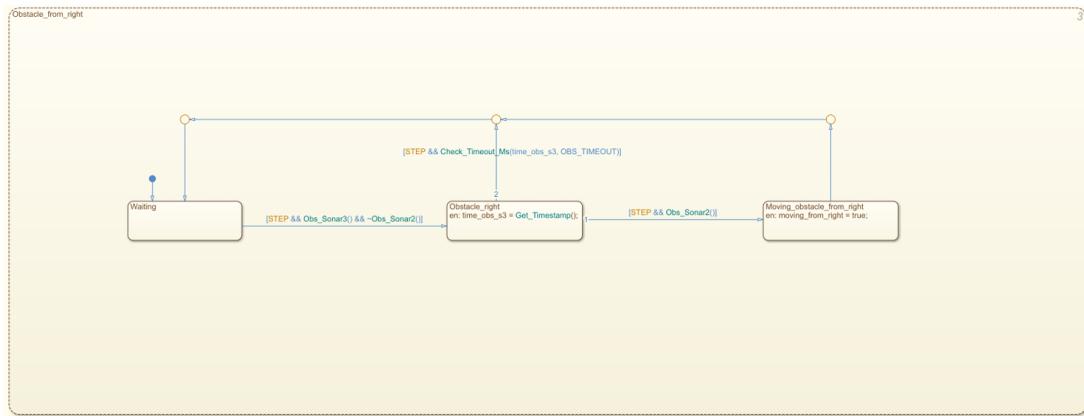


Figura 6.13: Obstacle\_from\_right

Nell'eventualità di una **rilevazione simultanea** proveniente da entrambe le direzioni, il sistema notifica allo **Stato Globale** una condizione di **criticità bilaterale**, che dovrà poi essere gestita in fase decisionale.

#### 6.4.2 Battery\_temperature\_manager

Questo automa ha il compito fondamentale di determinare se il rover ha la necessità di operare in **regime di velocità limitata** o se deve funzionare a **piene capacità**.

Per stabilire il regime operativo, l'algoritmo analizza i parametri di **voltaggio della batteria e temperatura** della **Board 1 (Slave)**. La logica interna ricalca una struttura a **tre automi paralleli con priorità crescente**, dove lo stato di limitazione prevale su quello nominale:

1. **Normal\_velocity**: è il primo blocco in esecuzione e definisce la condizione base. Il suo compito è asserire ciclicamente che il rover non sta operando in limitazione, impostando il regime di velocità **Nominale**. Essendo a **priorità minima**, questo stato è valido solo se nessun'altra condizione critica sovrascrive la decisione.

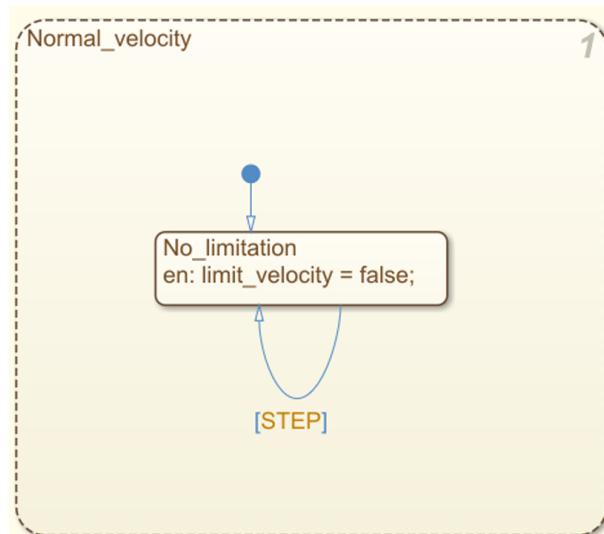


Figura 6.14: Normal\_velocity

2. **Battery\_manager**: valuta se la tensione della batteria monitorata è scesa al di sotto della soglia **LOW\_VOLTAGE**. In caso affermativo, impone che il sistema operi in **velocità limitata**. Questa condizione riflette uno stato in cui le prestazioni vengono ridotte forzatamente per preservare la carica residua, finché il voltaggio non rientri nella **soglia di sicurezza**.

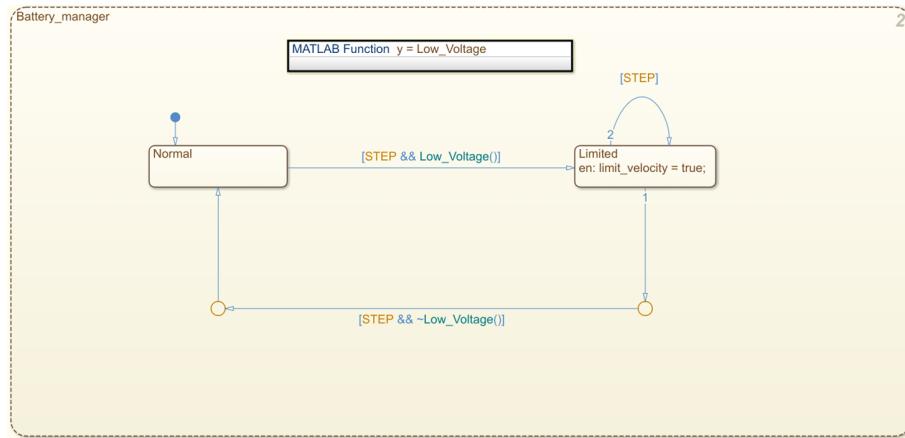


Figura 6.15: Battery\_manager

3. **Temperature\_manager**: verifica se la temperatura ha superato la soglia **HIGH\_TEMPERATURE** per un periodo continuativo superiore a **TEMP\_TIMEOUT** millisecondi. Se questa condizione di persistenza è soddisfatta, l'automa segnala che il rover deve operare in **velocità limitata**. Questo stato viene mantenuto finché la temperatura non rientra nelle **soglie di sicurezza**.

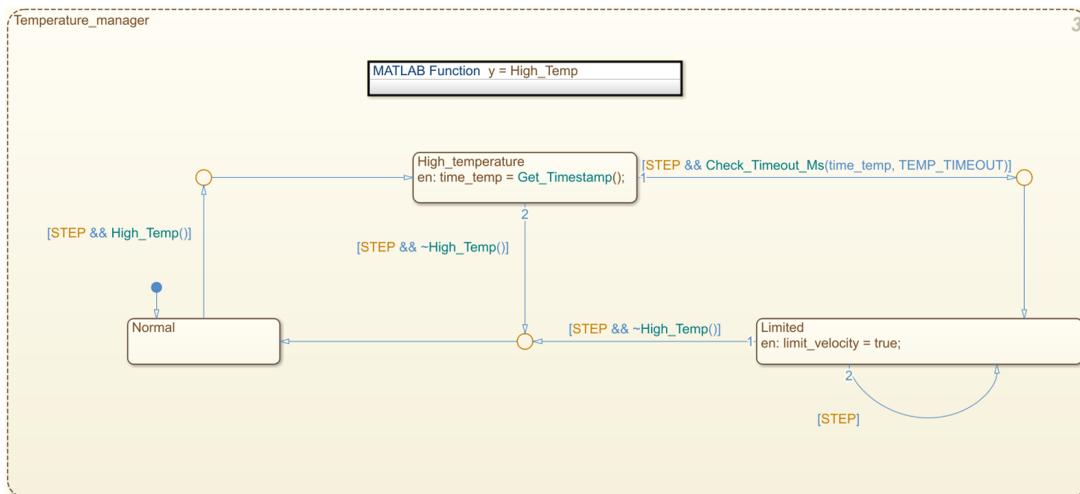


Figura 6.16: Temperature\_manager

### 6.4.3 Combo

Questo automa implementa una **logica sequenziale** finalizzata al riconoscimento di specifici **pattern di input** distribuiti su più cicli di esecuzione. Nello specifico, il modulo monitora l'attivazione in **successione temporale** dei due pulsanti di comando, interpretando l'ordine degli impulsi per discriminare **quattro distinte funzionalità operative**.

A garanzia della **robustezza del comando**, è introdotto un vincolo temporale di validità definito dal parametro **BUTTON\_TIMEOUT**. Una volta rilevata la pressione del primo tasto, si apre una **finestra temporale limitata** entro la quale deve giungere il secondo input; qualora il timeout scada senza il completamento della combinazione, l'automa invalida l'operazione e **resetta lo stato interno** tornando in **Waiting\_combo**.

Le quattro combinazioni ammissibili sono così codificate:

- **Decremento Velocità (Doppio click B1):** premendo il pulsante **B1** e, successivamente, ripremendo lo stesso **B1**, il sistema interpreta l'input come richiesta di **riduzione della velocità massima**. L'automa comanda un **decremento del parametro di saturazione** pari alla costante **VEL\_CHANGE**.
  - **Gestione Retromarcia Speciale (Sequenza B1 → B2):** l'attivazione sequenziale mista, innescata premendo prima **B1** e confermata premendo poi **B2**, inverte lo **stato logico** della modalità di **retromarcia speciale**. Allo startup del sistema, questa funzionalità è **abilitata di default**. Tale scelta progettuale è volta a massimizzare la **sicurezza**, garantendo che il rover operi inizialmente con le logiche di **inversione assistita**, mitigando il rischio di collisioni dovute a **punti ciechi dei sensori**.
  - **Incremento Velocità (Doppio click B2):** in modo speculare alla prima casistica, la ripetizione dell'input sul secondo tasto (prima **B2**, poi ancora **B2**) agisce come comando di **aumento della velocità massima**. L'automa comanda un **incremento del parametro di saturazione** pari alla costante **VEL\_CHANGE**.
  - **Controllo Rilevamento Ostacoli (Sequenza B2 → B1):** la combinazione incrociata finale, ottenuta premendo prima **B2** seguito da **B1**, interviene sui **parametri dei sistemi di protezione attiva**. Tale sequenza consente all'operatore di **ridurre la soglia di distanza dei sonar**, facilitando manovre volontarie in spazi confinati, o di **ripristinare la sensibilità standard**. All'inizializzazione, il rilevamento è **attivo di default**, assicurando che il sistema parta sempre nella condizione di **massima protezione anticollisione**.

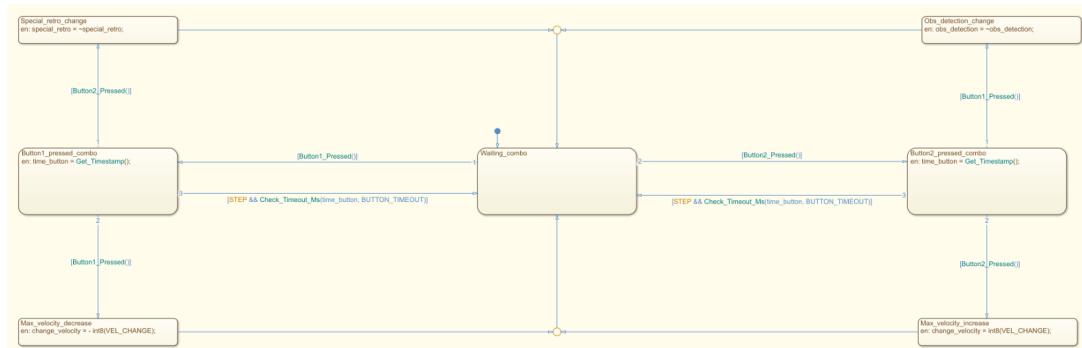


Figura 6.17: Combo

#### 6.4.4 Board\_decision

L'automa **Board\_decision** costituisce il nucleo decisionale del sistema, responsabile della generazione dei comandi finali verso i driver motori e i sottosistemi ausiliari.

Dal punto di vista architettonale, il macro-stato è organizzato come decomposizione parallela (AND-state). Sebbene le macchine a stati siano attive nello stesso passo di calcolo, la loro esecuzione segue un **ordine deterministico**: inizialmente viene eseguito il **Supervisore Diagnostico**, che aggiorna lo stato operativo globale, e successivamente vengono processati gli **Automi Decisionali** relativi ai canali di uscita (Relè, Multiplexer, Motori, Modalità di Guida e Illuminazione).

Segue l'analisi dei singoli moduli.

#### Working\_status\_manager

Il compito primario dell'automa è centralizzare il monitoraggio delle condizioni critiche hardware ed energetiche, astraendo la complessità della diagnostica in una singola variabile. Questa variabile funge da riferimento per tutti gli altri processi decisionali e attuativi, i quali adatteranno il proprio comportamento in base al contesto operativo definito qui. La logica di gestione operativa si articola su quattro stati sequenziali, progettati per garantire la stabilità delle letture all'avvio e la sicurezza durante il funzionamento:

- **Init\_working**: Rappresenta lo stato di ingresso. In questa fase preliminare, il sistema sospende le logiche di controllo attive per un numero di cicli predefinito (INIT\_COUNTER). Tale finestra di attesa è fondamentale per filtrare i transitori elettrici tipici dell'accensione (es. correnti di spunto dei motori o assestamento della tensione), evitando che fluttuazioni momentanee vengano erroneamente interpretate come guasti.
- **Normal\_working**: È la condizione di operatività standard, raggiunta se il rover risulta pienamente funzionale e privo di anomalie. In questo stato, il sistema opera a pieni parametri e non sussistono vincoli bloccanti.
- **Motor\_error\_working**: Lo stato viene attivato qualora la diagnostica rilevi guasti hardware critici ai driver o l'attivazione della "Combo di Panico". L'ingresso in questa modalità forza il sistema in una condizione di sicurezza incondizionata, inibendo le logiche di controllo standard.
- **Critical\_voltage\_working**: Se la tensione della batteria scende sotto la soglia CRITICAL\_VOLTAGE, l'automa transita in questo stato per preservare l'integrità delle celle. A differenza dell'errore motore, questa condizione è reversibile: il sistema prevede un percorso di ripristino automatico allo stato normale qualora la tensione si ristabilisse.

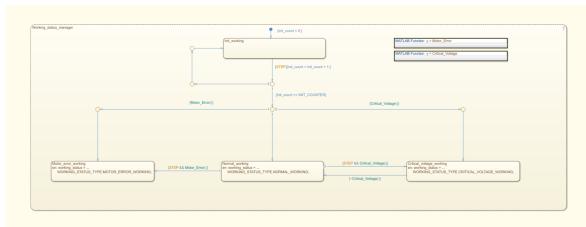


Figura 6.18: Working\_status\_manager

### Relay\_Manager

Questo modulo agisce come interblocco hardware di sicurezza per il sistema di trazione. La sua funzione è gestire l'abilitazione fisica dello stadio di potenza dei motori tramite il pilotaggio diretto del relè. La logica di controllo si articola su tre stati operativi distinti, progettati per garantire la sicurezza elettrica e meccanica del sistema in ogni condizione di funzionamento:

- **Funzionamento Nominale:** In assenza di anomalie, l'automa permane in questo stato e comanda la chiusura del relè. Questa azione garantisce la continuità verso i driver motori, condizione propedeutica necessaria affinché le decisioni di attuazione logica possano essere convertite in movimento fisico.
- **Gestione Errori Critici:** Qualora venga rilevato un guasto critico ai motori, il sistema forza immediatamente l'apertura del relè. Tale azione disaccoppia fisicamente l'alimentazione di potenza, provocando un arresto inerziale non controllato ma sicuro. Questa transizione è irreversibile: una volta entrati in questo stato di protezione, non è previsto un ritorno automatico alla condizione normale senza un reset del sistema.
- **Protezione Energetica:** Questa logica interviene se la tensione di alimentazione scende sotto la soglia di guardia. Anche in questo caso, l'automa comanda l'apertura del relè per proteggere le celle della batteria da scariche sotto carico. A differenza dell'errore motore, questa condizione è reversibile: se la condizione di sottotensione viene meno, il sistema è autorizzato a richiudere il relè e ripristinare l'operatività nominale.

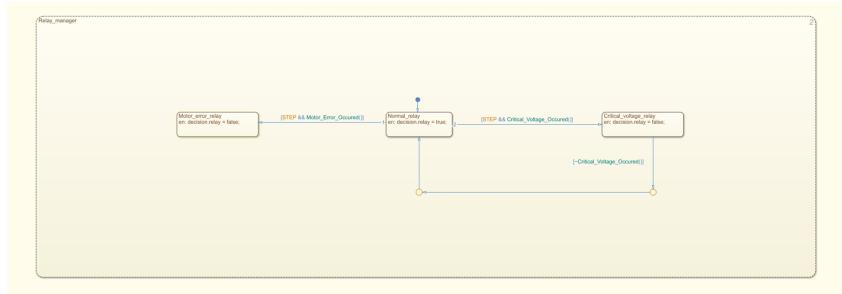


Figura 6.19: Relay\_manager

### Mux\_Manager

Questo modulo governa la logica di selezione della sorgente di comando per i motori (Multiplexer) e definisce la strategia di regolazione (Ciclo Aperto e Ciclo Chiuso). Il suo scopo è garantire la continuità del moto e prevenire transitori bruschi durante le commutazioni tra funzionamento nominale e degradato. La logica si alterna tra due stati operativi principali:

1. **Funzionamento Nominale (Normal\_mux):** In condizioni operative standard, il segnale di selezione del Multiplexer è disattivo. In questa configurazione, il controllo dei motori è affidato alla logica di default (lo Slave controlla i motori).
2. **Gestione Modalità Degradata (Degraded\_mux):** Qualora il supervisore diagnostico transiti nello stato Degraded, l'automa entra in questo stato e forza preventivamente i flag di controllo open\_loop a true. Nota Bene: Questi valori non

sono destinati all'uso immediato durante l'avaria, ma servono a preparare la fase di rientro. Quando la comunicazione viene ristabilita e la board torna in stato Normal, l'automa permane intenzionalmente in questo stato, mantenendo il controllo in Ciclo Aperto (Open Loop). Questo accorgimento è fondamentale per allineare le dinamiche del sistema ed evitare che il controllo a ciclo chiuso si riattivi bruscamente mentre il veicolo è ancora in moto.

Il ritorno alla condizione nominale non avviene automaticamente al solo ripristino dello stato Normal della board. Per garantire un passaggio smooth dal controllo in anello aperto a quello in anello chiuso, il sistema impone una condizione di sicurezza aggiuntiva:

- **Check Veicolo Fermo:** La transizione verso Normal\_mux è abilitata solo se il rover è fermo. Questo vincolo impedisce che il loop di controllo venga riagganciato con il veicolo in movimento, situazione che potrebbe causare instabilità.
- **Reset del PID (Specificità Slave):** Contestualmente alla transizione di ripristino, viene triggerata l'esecuzione della funzione di reset del PID. Questa operazione, esclusiva dell'architettura Slave, è mandatoria per azzerare la memoria del termine integrale accumulata durante la fase degradata. Tale accorgimento neutralizza il rischio di wind-up e di bump, garantendo che il loop di controllo riprenda la regolazione partendo da condizioni iniziali neutre.

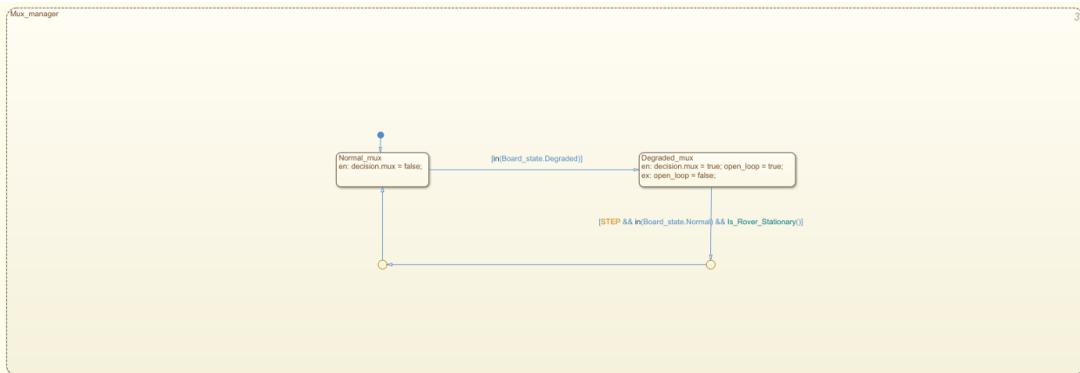


Figura 6.20: Mux\_Manager

### Routine\_Manager

Questo modulo rappresenta il **cuore algoritmico** del sistema di trazione. La sua responsabilità è calcolare i riferimenti puntuali di velocità (RPM) per i singoli motori e definire la strategia di frenata. Architetturalmente, il macro-stato implementa una **decomposizione parallela (AND-state)** che coordina due processi:

- **Max\_velocity\_handler:** Questo automa opera come supervisore della velocità massima del veicolo. Oltre a processare i comandi utente per la regolazione della velocità, esso definisce la soglia di saturazione del sistema. Il modulo monitora costantemente le condizioni operative: qualora la velocità massima impostata ecceda i limiti di sicurezza correnti, il valore viene automaticamente **saturato** alla soglia ammissibile, garantendo che i riferimenti inviati ai motori rimangano sempre all'interno dei margini di sicurezza.

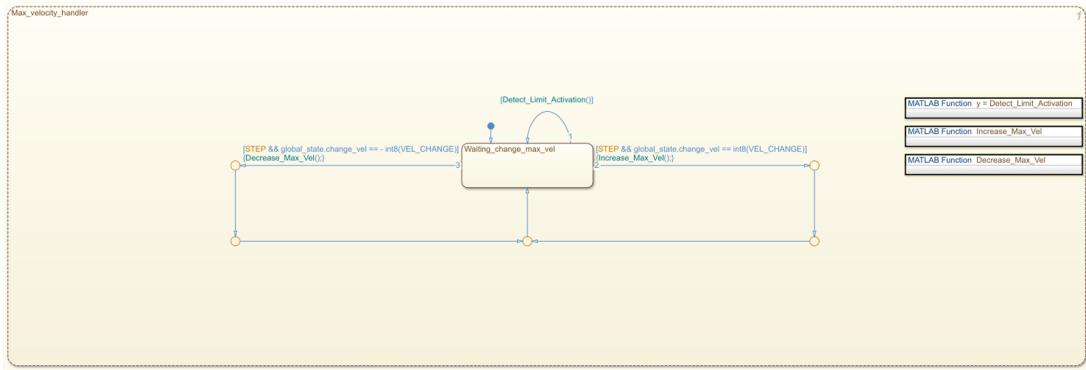


Figura 6.21: Max\_velocity\_handler

- **Compute\_routine:** Questo automa costituisce il **nucleo esecutivo** del controllo di trazione. La sua responsabilità primaria è la traduzione delle intenzioni di guida e delle procedure di sicurezza in comandi fisici per gli attuatori, assicurando che nessuna anomalia hardware possa compromettere l'integrità del veicolo. L'architettura interna è gerarchica e stabilisce una **priorità** tra la sicurezza operativa e il calcolo cinematico:
  1. **Calcolo Nominale (Normal\_routine):** In condizioni operative standard, il sistema risiede stabilmente in questo stato. Qui avviene l'elaborazione ciclica dell'algoritmo cinematico, che trasforma gli input di direzionalità in riferimenti di velocità per le singole ruote.
  2. **Protezione Hardware (Motor\_error\_routine):** Questo stato viene attivato all'intercettazione di guasti critici ai motori. La logica di protezione impone l'immediato **azzeramento dei riferimenti di velocità** e l'attivazione della **frenata di emergenza**. Data la gravità del guasto, questo stato è bloccante: il sistema non può tornare autonomamente al funzionamento normale fino a un reset completo dell'architettura.
  3. **Protezione Energetica (Critical\_voltage\_routine):** Interviene qualora la tensione della batteria scenda sotto la soglia di guardia. Analogamente al caso di guasto motori, il sistema forza i riferimenti a zero e comanda la frenata di emergenza per rimuovere istantaneamente il carico elettrico. A differenza del precedente, questo stato è **reversibile**: prevede un percorso di ripristino che autorizza il ritorno alla Normal\_routine e la riabilitazione del moto se la condizione di allarme rientra.

Esternamente al blocco logico, è definita una condizione di guardia globale associata alla modalità degradata della board. L'attivazione di questa condizione provoca il **reset istantaneo** dell'intera logica di calcolo automatico. Questa misura preventiva assicura che, al momento del ripristino della comunicazione, l'algoritmo riparta da uno stato neutro, evitando l'esecuzione di comandi residui.

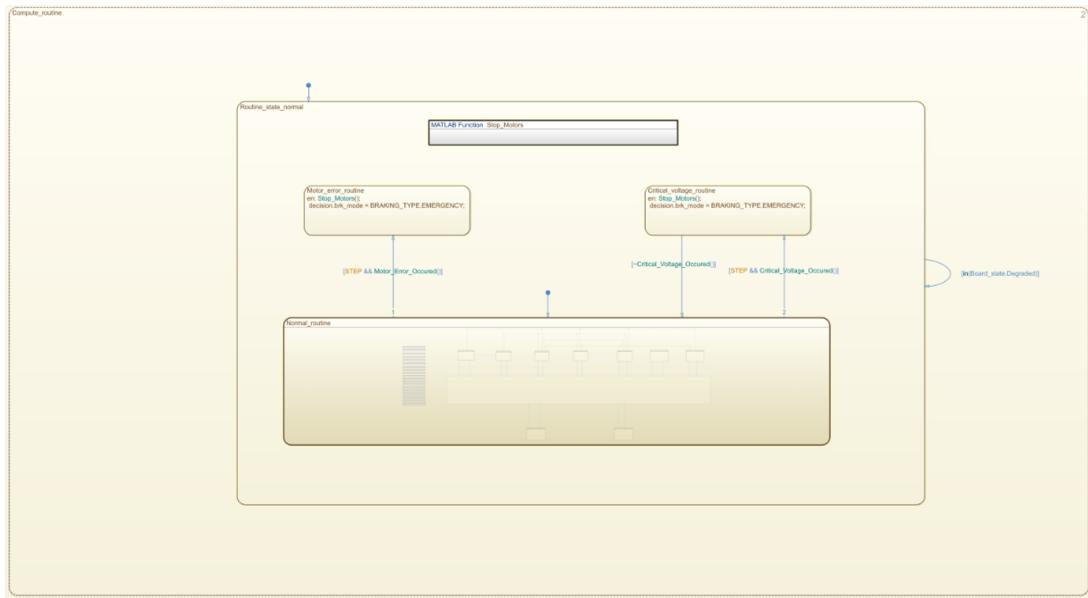


Figura 6.22: Compute\_routine

Esaminiamo ora nel dettaglio l'architettura interna dello stato Normal\_routine, analizzando la logica che governa la selezione e l'attivazione delle diverse strategie di controllo cinematico.

**Normal\_routine** In condizioni operative nominali, l'automa si occupa di selezionare la Routine di comportamento più appropriata secondo una gerarchia decrescente di priorità. Di seguito sono descritte le routine implementate, ordinate dalla priorità massima (sicurezza critica) alla minima (controllo ordinario).

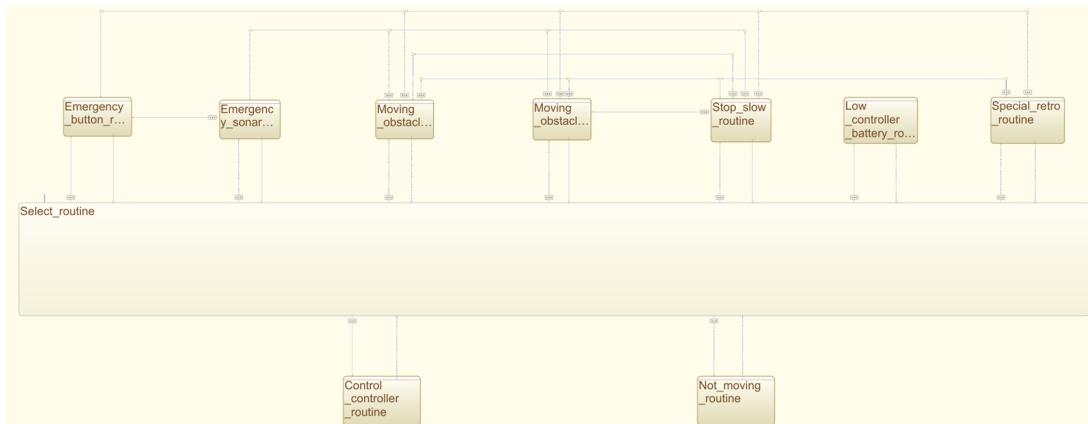


Figura 6.23: Normal\_routine

1. **Emergency\_button\_routine** (Arresto Manuale - Priorità Massima): Questa routine gestisce l'attivazione del pulsante di emergenza fisico. La sua esecuzione è subordinata alla validità del segnale del controller (batteria del controller > LOW\_CONTROLLER\_BATTERY).

Essendo il livello più alto della gerarchia di sicurezza, questa condizione prevale incondizionatamente su qualsiasi altra logica (inclusa l'evasione ostacoli). Al suo verificarsi, i riferimenti di velocità vengono immediatamente azzerati e la modalità

di frenata viene forzata su emergenza, garantendo un arresto istantaneo e prioritario del veicolo.

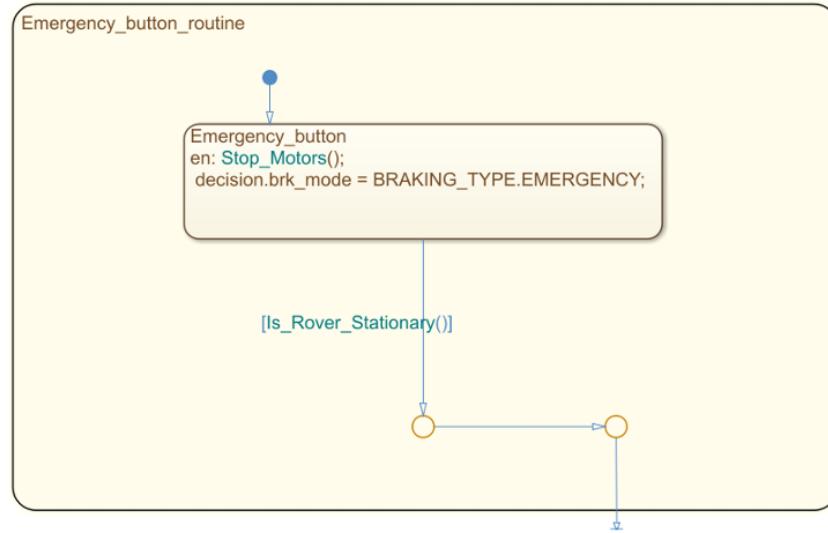


Figura 6.24: Emergency\_button\_routine

2. **Emergency\_sonar\_routine**: Si attiva quando il rover è in moto di avanzamento e almeno uno dei sonar rileva un ostacolo a distanza critica, prefigurando un impatto imminente.

- Fase 1 (Arresto): Azzeramento immediato della velocità e attivazione frenata di emergenza.
- Fase 2 (Evasione): Successivamente all'arresto, il sistema valuta lo spazio laterale. Se viene individuato un varco, il rover esegue una rotazione controllata verso il lato libero. Qualora entrambi i lati risultino ostruiti, il veicolo permane in posizione di fermo. In caso di entrambi i lati liberi, il sistema risolve l'ambiguità applicando una preferenza di default per la manovra a destra. Una volta determinata la via di fuga, il sistema verifica l'affidabilità del giroscopio (gyroError).
  - Rotazione in Anello Chiuso: Se il giroscopio è attivo e funzionante, il sistema entra nello stato Turn\_right/left\_gyro. La manovra è controllata in feedback: il rover ruota finché l'integrazione del segnale giroscopico non raggiunge l'angolo target (TURN\_ANGLE). È presente una condizione di timeout (TURN\_LIMIT\_COUNT) per prevenire stalli infiniti qualora l'angolo non venisse raggiunto.
  - Rotazione in Anello Aperto: In caso di guasto rilevato al giroscopio, il sistema commuta automaticamente sulla strategia Turn\_right/left\_no\_gyro. In questa modalità degradata, la rotazione non è basata sull'angolo reale ma su una stima temporale precalcolata (TURN\_COUNT), garantendo l'esecuzione della manovra evasiva anche in assenza di feedback.

Al termine della rotazione, il sistema transita nello stato di stop dedicato (Stop\_rotation), azzerando i riferimenti motori e preparandosi a riprendere la navigazione.

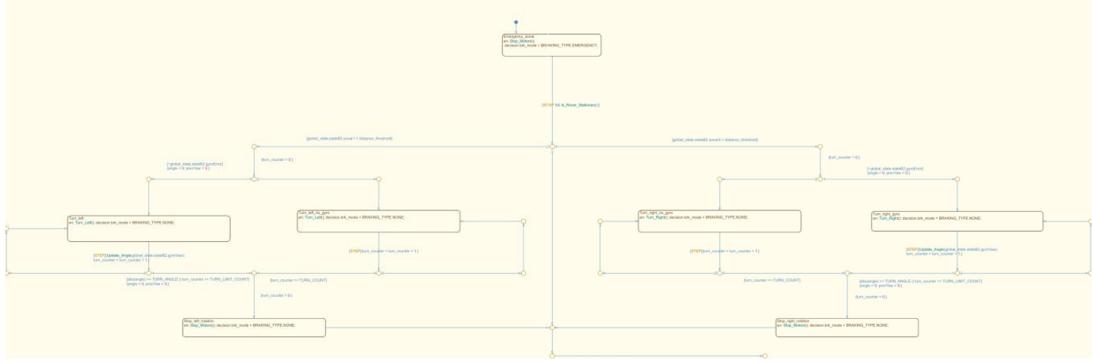


Figura 6.25: Emergency\_sonar\_routine

3. **Moving\_obstacle\_routine:** L'attivazione di questa logica è vincolata alla fase di marcia in avanti e alla contestuale presenza del flag di rilevamento sonar. L'intervento avviene puntualmente qualora lo Stato Globale segnali la presenza di un ostacolo dinamico, precedentemente classificato dagli automi di rilevamento:

- **Caso Destro (Moving\_obstacle\_from\_right):** In caso di rilevamento di ostacolo dinamico proveniente da destra, il sistema attiva una manovra di **evasione dinamica**. Viene generata una traiettoria curvilinea verso destra che combina la velocità di avanzamento corrente con una componente di sterzata. Questo permette al rover di deviare fluidamente dalla linea di collisione senza interrompere la marcia.

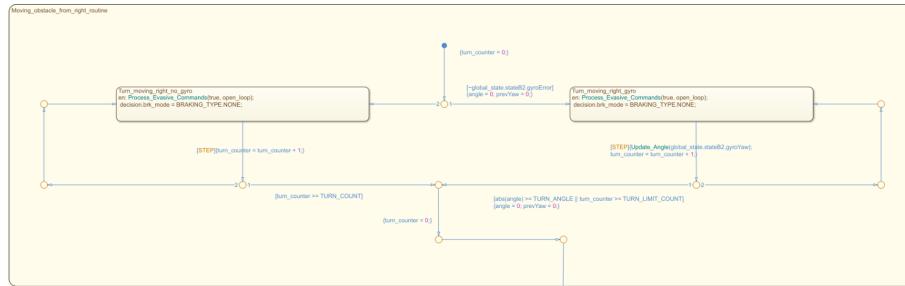


Figura 6.26: Moving\_obstacle\_from\_right

- **Caso Sinistro (Moving\_obstacle\_from\_left):** Logica speculare al caso destro. A fronte del rilevamento, il rover intraprende una manovra di evasione dinamica verso sinistra.

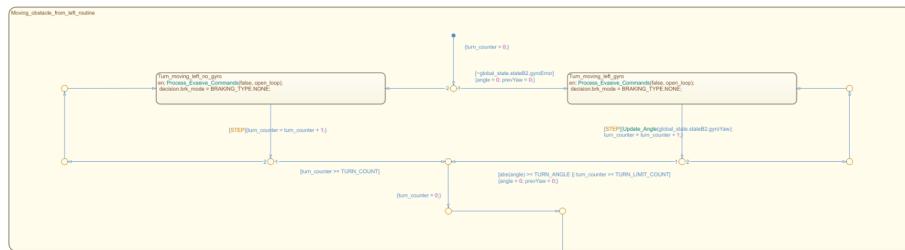


Figura 6.27: Moving\_obstacle\_from\_left

4. **Stop\_slow\_routine:** Questa routine impone un arresto controllato del veicolo (frenata smooth). La logica di attivazione è composita e sorveglia tre scenari distinti:

- Sicurezza Attiva (Ostacoli): Interviene durante il moto in avanti (con flag di rilevamento sonar attivo) se si verifica una delle seguenti condizioni:
  - Il sonar centrale rileva un ostacolo statico entro la soglia di guardia ( $d \leq \text{MAX\_DISTANCE}$ ).
  - Il sistema rileva simultaneamente ostacoli in movimento provenienti da entrambi i lati (MOVING\_FROM\_BOTH). In questo scenario, dove una manovra evasiva laterale sarebbe inefficace, il rover opta per l'arresto.
- Interazione Utente: Attivata dalla pressione volontaria del pulsante di Stop sul controller, a condizione che il livello di batteria del radiocomando sia sufficiente a garantire l'affidabilità del segnale.
- Failsafe Connettività: La routine scatta automaticamente anche in caso di Errore del Controller. Se viene persa la comunicazione o rilevata un'anomalia nel dispositivo di input, il sistema porta il veicolo in uno stato di arresto per prevenire comportamenti imprevedibili.

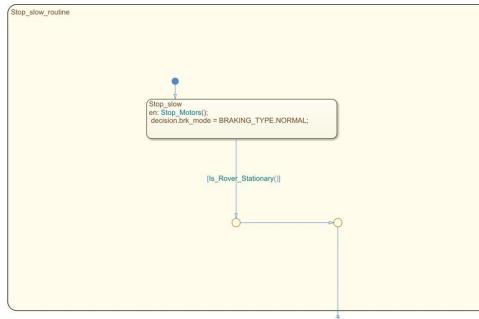


Figura 6.28: Stop\_slow\_routine

5. **Low\_controller\_battery\_routine** si attiva quando la telemetria segnala che il livello di carica della batteria del controller remoto è sceso al di sotto della soglia LOW\_CONTROLLER\_BATTERY. In tale condizione, il sistema azzerà immediatamente i riferimenti di velocità e imposta la modalità di frenata *smooth*, realizzando un arresto controllato volto a prevenire comportamenti imprevedibili dovuti alla perdita di affidabilità del dispositivo di comando.

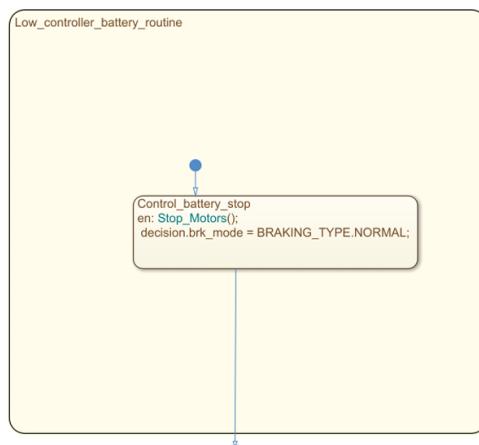


Figura 6.29: Low\_controller\_battery\_routine

**6. Special\_retro\_routine:** Questa routine supervisiona l'esecuzione di una manovra automatizzata di retromarcia speciale. L'attivazione è vincolata alla coesistenza di tre condizioni logiche rigorose:

- Integrità del Controllo: Il sistema deve operare in modalità Closed Loop (escludendo quindi le condizioni "Open Loop" nelle fasi di ripristino della comunicazione).
- Abilitazione Modale: Deve essere attivo il flag spc\_retro.
- Intentionalità Utente: L'operatore deve impartire esplicitamente un comando di retromarcia tramite lo stick (controller\_y < CONTROLLER\_ZERO).

Una volta innescata, la routine gestisce la manovra secondo il seguente flusso sequenziale:

Strategia di Controllo (Gyro vs Timer): Il sistema discrimina la legge di controllo basandosi sulla salute della sensoristica inerziale:

- Turn\_back\_gyro: Se il giroscopio è affidabile, il rover esegue la rotazione monitorando l'angolo reale. La transizione di uscita avviene al raggiungimento del target TURN\_BACK\_ANGLE o, in via cautelativa, per intervento di un timeout di sicurezza (TURN\_BACK\_LIMIT\_COUNT).
- Turn\_back\_no\_gyro: In caso di avaria del giroscopio, la logica commuta su una strategia a tempo. La rotazione viene arrestata una volta trascorsa la finestra temporale TURN\_BACK\_COUNT.

Al termine della fase attiva, il sistema transita nello stato di attesa Stop\_back\_rotation, imponendo la frenata e l'azzeramento dei riferimenti. Per uscire dalla routine e restituire il controllo alla guida manuale, è implementato un meccanismo di sicurezza: l'utente è obbligato a rilasciare lo stick (portando controller\_y a zero). Questa condizione di reset previene il riavvio ciclico e involontario della manovra di retromarcia automatica.

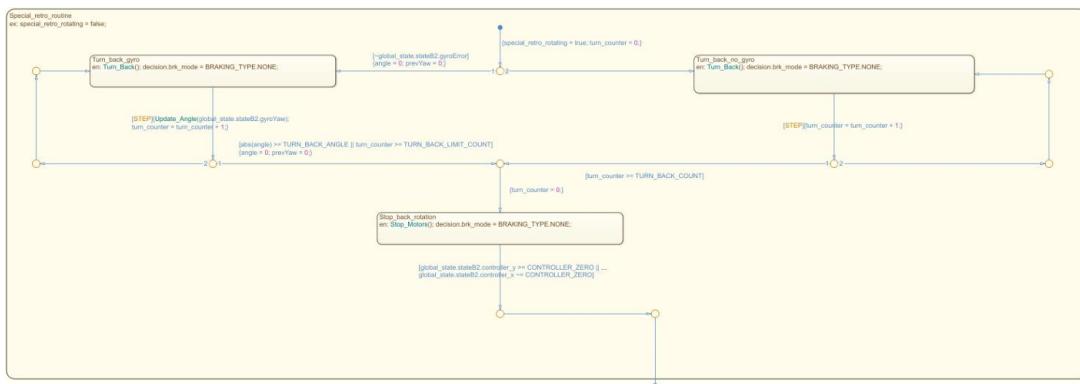


Figura 6.30: Special\_retro\_routine

**7. Not\_moving\_routine:** Questa routine funge da barriera di sicurezza logica. Quando il veicolo è fermo e viene rilevato un ostacolo frontale critico ( $d \leq MAX\_DISTANCE$ ), il sistema valuta l'input del pilota. Se viene comandato un avanzamento (controller\_y > 0) mentre il rilevamento ostacoli è attivo, la routine

forza i driver motori a valore nullo, mantenendo il veicolo in stato di arresto e prevenendo l'impatto.

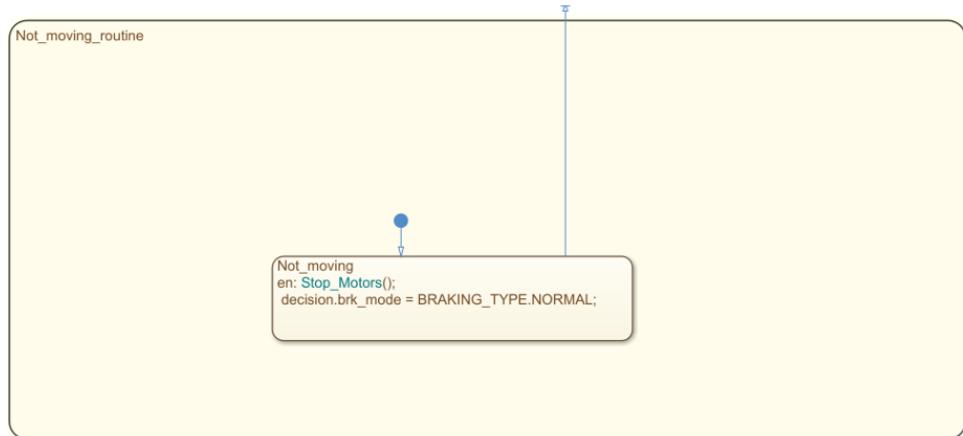


Figura 6.31: Not\_moving\_routine

8. **Control\_controller\_routine:** Rappresenta lo stato di funzionamento ordinario, attivo quando nessuna delle condizioni di emergenza o sicurezza precedenti è verificata. Il sistema elabora gli input del joystick (avanzamento e sterzata) trasformandoli in riferimenti di velocità per i singoli motori.

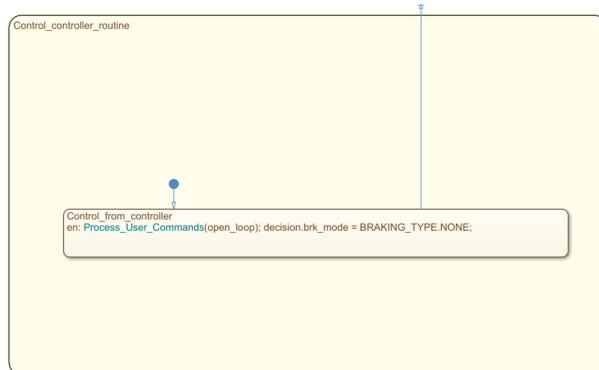


Figura 6.32: Control\_controller\_routine

### Mode\_manager

Questo modulo parallelo definisce il profilo di guida attivo, determinando la "reattività" dinamica del rover. La logica di controllo integra la selezione manuale dell'operatore con meccanismi di valutazione dello stato di salute del sistema. Il funzionamento si divide in due aree logiche:

- **Gestione Manuale dei Profili (Normal\_driving):** In condizioni operative standard, l'automa risiede all'interno di questo macro-stato, dove la selezione della modalità è affidata al controllo dell'utente. Ad ogni pressione del pulsante dedicato, il sistema commuta sequenzialmente tra tre configurazioni predefinite:
  - Mode\_DEFAULT: È lo stato di avvio e rappresenta il profilo bilanciato per l'utilizzo generico.

- Mode\_SPORT: Profilo orientato alle prestazioni, caratterizzato da dinamiche più aggressive.
  - Mode\_ECO: Profilo conservativo, ottimizzato per il risparmio energetico.- **Gestione della Diagnostica:** L'automa monitora costantemente i flag di errore globali e può forzare autonomamente il cambio di modalità per garantire la sicurezza:
  - Fallback per Guasto Motori (Motor\_error\_driving): Qualora venga rilevato un errore critico dei motori, il sistema abbandona la logica di selezione manuale e transita in questo stato di sicurezza. Qui viene forzata l'impostazione DEFAULT, considerata la configurazione neutra e più prevedibile per gestire l'anomalia. Questa transizione è irreversibile all'interno del ciclo di accensione.
  - Protezione Energetica (Critical\_voltage\_driving): Se la tensione della batteria scende sotto la soglia critica, il manager interviene forzando la modalità ECO. Questa azione mira a ridurre l'assorbimento di corrente limitando le prestazioni del veicolo, nel tentativo di preservare la carica residua. A differenza del guasto motori, questa condizione è reversibile: se la tensione si ristabilisce, il sistema restituisce il controllo alla logica manuale.

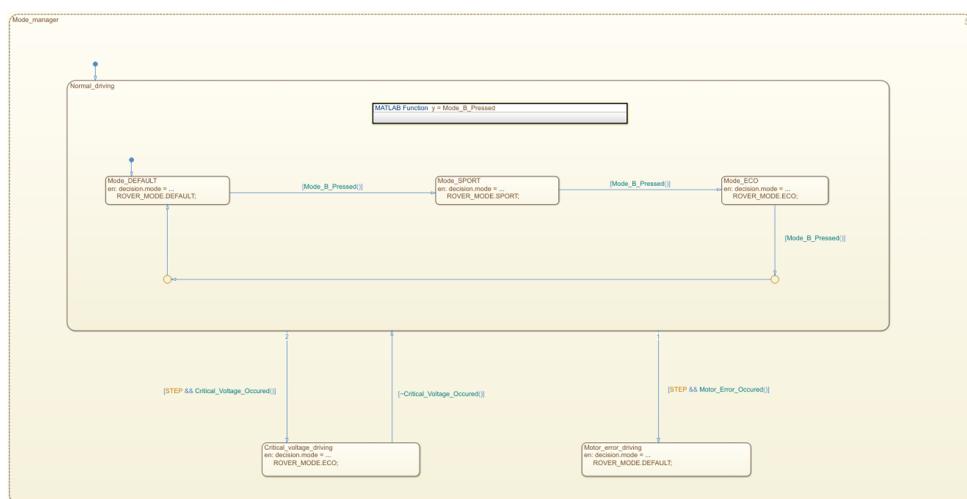


Figura 6.33: Mode\_manager

### Lights\_Manager

Questo modulo parallelo gestisce il feedback visivo del rover, orchestrando lo stato dei LED frontali, della fanaleria posteriore e dell'insegna posteriore. La logica di controllo incrocia tre livelli decisionali: la selezione manuale dell'utente, la cinematica del veicolo e lo stato di salute del sistema.

**Gestione delle Modalità Operative** Il nucleo della macchina a stati regola l'abilitazione funzionale del sistema di illuminazione. Attraverso la pressione ciclica del pulsante dedicato, l'utente naviga tra tre configurazioni:

1. **Modalità OFF**: Il sottosistema di illuminazione è forzato allo spegnimento completo. Nessun segnale viene emesso, indipendentemente dal movimento del veicolo.
2. **Modalità ON**: Le luci sono abilitate in modalità continua, anche a veicolo immobile.
3. **Modalità AUTO**: Replica le funzionalità dinamiche della modalità ON, ma quando il rover è fermo, i fari anteriori vengono spenti.

**Logica di Attuazione Dinamica** lo stato dei led viene calcolato dalla funzione Update\_Rover\_Lights, che implementa un arbitraggio rigoroso basato su una scala di priorità decrescente. Il sistema valuta le condizioni cinematiche dalla più critica alla meno prioritaria:

1. **Rotazione Speciale (Priorità 1 - Massima)**: Il sistema sovrascrive ogni altra segnalazione attivando un lampeggio bianco sincrono su entrambi i lati anteriori e configurando le luci posteriori su un pattern dedicato.
2. **Frenata Attiva (Priorità 2)**: Vengono accesi i fari bianchi fissi anteriori e le luci di stop posteriori.
3. **Manovre Direzionali (Priorità 3 e 4)**: Il sistema monitora sia le rotazioni sul posto (Pivot) sia le curve in avanzamento (Sterzata). Sebbene le logiche di rilevamento siano distinte, esse producono il medesimo feedback visivo: l'attivazione del lampeggio rosso sul lato di svolta e della relativa freccia direzionale posteriore.
4. **Retromarcia (Priorità 5)**: I fari anteriori restano bianchi mentre posteriormente si attivano le luci bianche di retromarcia.
5. **Marcia Avanti (Priorità 6)**: Il rover mantiene l'assetto standard: fari bianchi anteriori e luci di posizione rosse posteriori.
6. **Stazionamento (Priorità 7 - Minima)**: A veicolo fermo, il comportamento dipende dalla modalità selezionata (ON o AUTO), determinando se mantenere i fari anteriori accesi o spegnerli.

Indipendentemente dalla cinematica istantanea, il pannello LED posteriore mappa il colore alla modalità di guida selezionata:

- Bianco: Modalità DEFAULT (Guida normale).
- Arancione: Modalità SPORT (Prestazioni elevate).
- Verde: Modalità ECO (Risparmio energetico).

**Gestione delle Anomalie** Le logiche descritte sopra sono subordinate allo stato di salute globale del rover. Due condizioni di guardia possono forzare l'uscita dagli stati operativi:

- Errore Motori: In caso di errore dei motori, il sistema transita nello stato Motor\_error\_lights. Qui viene imposta una segnalazione di emergenza visiva:
  - Led Anteriori: Rosso lampeggiante lento.
  - Led Posteriori: Luci di emergenza.
  - Cartello Posteriore: Rosso fisso.
- Criticità Energetica: Se la tensione della batteria scende sotto la soglia di guardia, il manager transita in Critical\_voltage\_lights e forza lo spegnimento totale delle luci per preservare la carica residua della batteria.

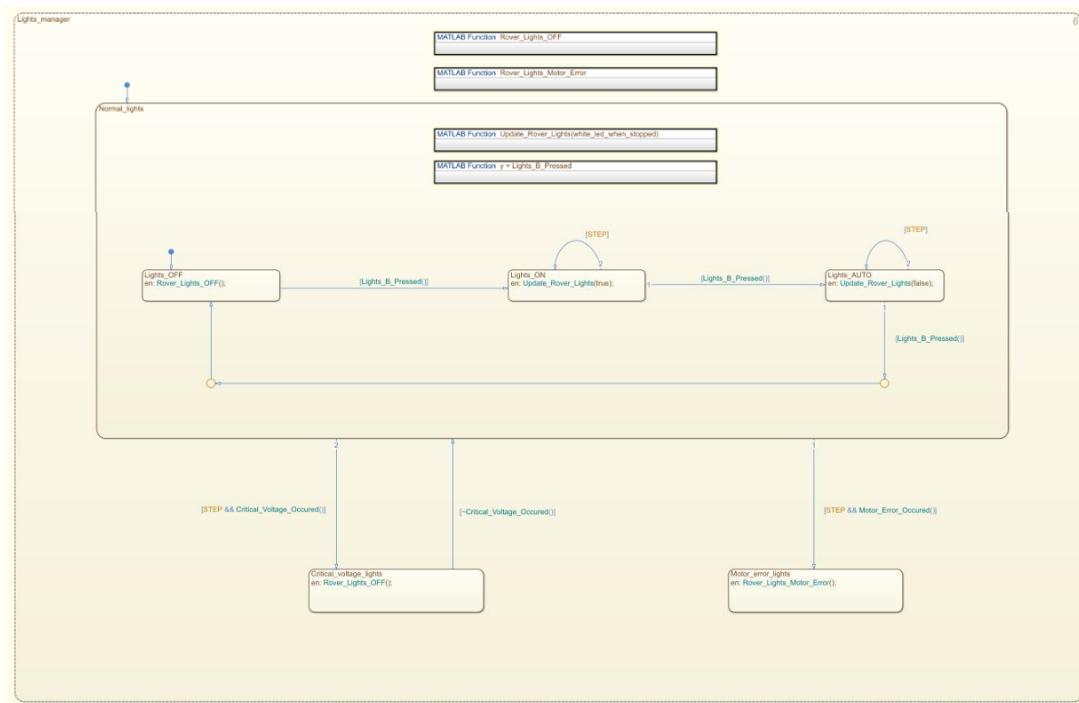


Figura 6.34: lights\_manager

# Generazione del codice

La logica a stati finiti, modellata tramite diagrammi Stateflow in ambiente Simulink, viene tradotta in firmware eseguibile mediante l'utilizzo di **Embedded Coder**.

Embedded Coder consente la conversione automatica del modello in codice C/C++ ottimizzato e conforme agli standard industriali. Il processo di generazione garantisce la corrispondenza tra il comportamento simulato e quello eseguito sul microcontrollore, riducendo drasticamente il rischio di errori introdotti dalla codifica manuale.

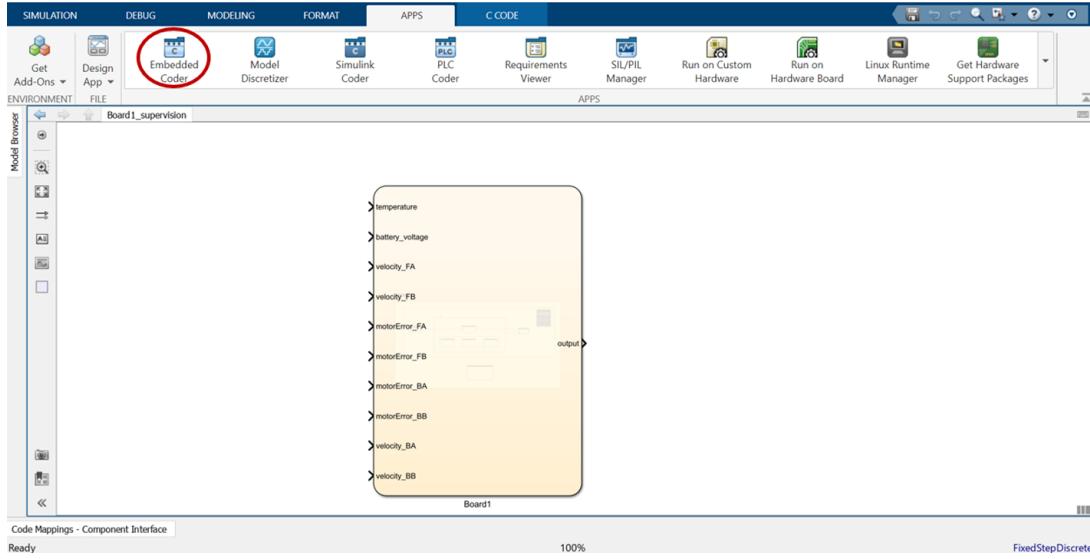


Figura 7.1: Scelta Embedded Coder

## 7.1 Corrispondenza tra tipi e occupazione in memoria

Durante il processo di generazione automatica del codice, i tipi di dato definiti nel modello Simulink/Stateflow vengono tradotti nei corrispondenti tipi C in funzione della configurazione del target embedded. Nel caso in esame, il sistema è basato su un'architettura ARM a 32 bit; di conseguenza, la rappresentazione dei tipi primitivi segue le convenzioni definite dai file di supporto generati da Embedded Coder (ad esempio `rtwtypes.h`).

La Tabella 7.1 riporta la corrispondenza tra i tipi utilizzati nel modello e la loro rappresentazione nel codice C generato, insieme alla relativa occupazione teorica in memoria.

Tipo MATLAB	Tipo C generato	Occupazione (byte)
single	real32_T (float)	4
uint16	uint16_T	2
int16	int16_T	2
uint8	uint8_T	1
int8	int8_T	1
boolean	boolean_T	1
Enumerazione	uint8_T	1

Tabella 7.1: Mapping tra tipi MATLAB e tipi C generati

La definizione dei `Simulink.Bus` nel modello comporta la generazione automatica di strutture `struct` nel codice C. È quindi possibile stimare l'occupazione teorica delle principali strutture di stato sommando le dimensioni dei singoli campi.

### 7.1.1 Stima dell'occupazione — StateBusB1

L'occupazione teorica del bus `StateBusB1` è stimabile come segue:

- 2 variabili di tipo `single`  $\rightarrow 2 \times 4 = 8$  byte
- 4 variabili di tipo `int16`  $\rightarrow 4 \times 2 = 8$  byte
- 4 variabili di tipo `boolean`  $\rightarrow 4 \times 1 = 4$  byte

Occupazione teorica minima:

$$8 + 8 + 4 = 20 \text{ byte}$$

### 7.1.2 Stima dell'occupazione — StateBusB2

Per il bus `StateBusB2`:

- 1 variabile di tipo `single`  $\rightarrow 4$  byte
- 3 variabili di tipo `uint16`  $\rightarrow 3 \times 2 = 6$  byte
- 2 variabili di tipo `uint16` (assi controller)  $\rightarrow 2 \times 2 = 4$  byte
- 6 variabili di tipo `boolean` (pulsanti)  $\rightarrow 6 \times 1 = 6$  byte
- 1 variabile di tipo `uint8`  $\rightarrow 1$  byte
- 2 variabili di tipo `boolean` (error flag)  $\rightarrow 2$  byte

Occupazione teorica minima:

$$4 + 6 + 4 + 6 + 1 + 2 = 23 \text{ byte}$$

### 7.1.3 Stima dell'occupazione — **GsBus**

Per il bus **GsBus**:

- 1 variabile di tipo Bus StateBusB1 → 20 byte
- 1 variabile di tipo Bus StateBusB2 → 23 byte
- 1 variabile di tipo enumerativo → 1 byte
- 3 variabili di tipo boolean → 3 byte
- 1 variabile di tipo int8 → 1 byte

Occupazione teorica minima:

$$20 + 23 + 1 + 3 + 1 = 48 \text{ byte}$$

### 7.1.4 Stima dell'occupazione — **DecBus**

Per il bus **DecBus**:

- 4 variabili di tipo single →  $4 \times 4 = 16$  byte
- 6 variabili di tipo enumerativo →  $6 \times 1 = 6$  byte
- 2 variabili di tipo boolean → 2 byte

Occupazione teorica minima:

$$16 + 6 + 2 = 24 \text{ byte}$$

### 7.1.5 Considerazioni sull'allineamento

Le stime riportate rappresentano l'occupazione teorica minima, ottenuta come somma delle dimensioni dei singoli campi. Tuttavia, su architetture a 32 bit il compilatore può introdurre meccanismi di allineamento (padding) per garantire un accesso efficiente alla memoria, in particolare per variabili di tipo floating-point o per campi con dimensione superiore al byte.

Di conseguenza, l'occupazione effettiva delle strutture **struct** generate può risultare superiore rispetto alla stima teorica calcolata.

L'analisi della rappresentazione in memoria consente di collegare in modo diretto la modellazione ad alto livello effettuata in Simulink/Stateflow con l'implementazione firmware sul target embedded, evidenziando l'impatto delle scelte di tipizzazione sull'utilizzo delle risorse hardware.

## 7.2 Requisiti preliminari alla generazione

Affinché il processo di generazione produca codice corretto e coerente con l'architettura firmware, è necessario soddisfare alcuni vincoli preliminari.

### 7.2.1 Definizione rigorosa dei tipi di dato

Una condizione indispensabile per la generazione del codice è la corretta definizione dei tipi.

Per garantire la coerenza tra i segnali Simulink e le strutture C generate, sono stati utilizzati **Simulink Bus Objects**, definiti tramite il Type Editor di MATLAB. Per ciascun elemento è stato specificato esplicitamente il tipo di dato.

Le definizioni dei bus sono state salvate nel file:

`board_workspace.mat`

Prima di ogni compilazione o simulazione, tale file deve essere caricato nel *Base Workspace*, in modo da rendere disponibili le definizioni necessarie al generatore di codice.

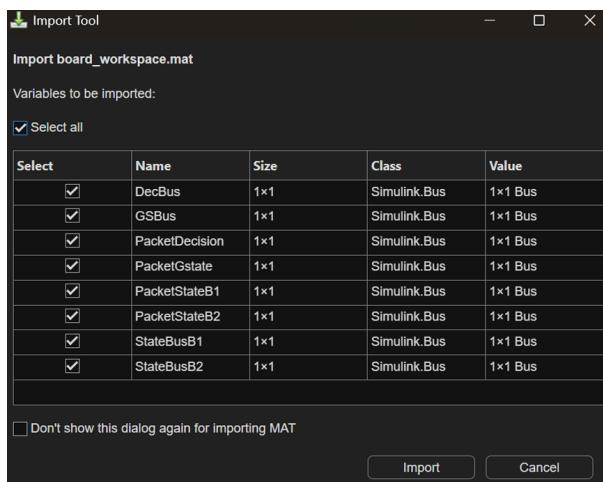


Figura 7.2: Import del workspace

### 7.2.2 Configurazione diagnostica

Per evitare discrepanze tra segnali e strutture dati, è stato configurato il parametro:

`Element name mismatch → error`

Questa impostazione obbliga Simulink a verificare la perfetta corrispondenza tra i nomi dei segnali nel modello e i campi dei Bus Objects.

In assenza di questa configurazione, il codice potrebbe risultare sintatticamente corretto ma semanticamente errato.

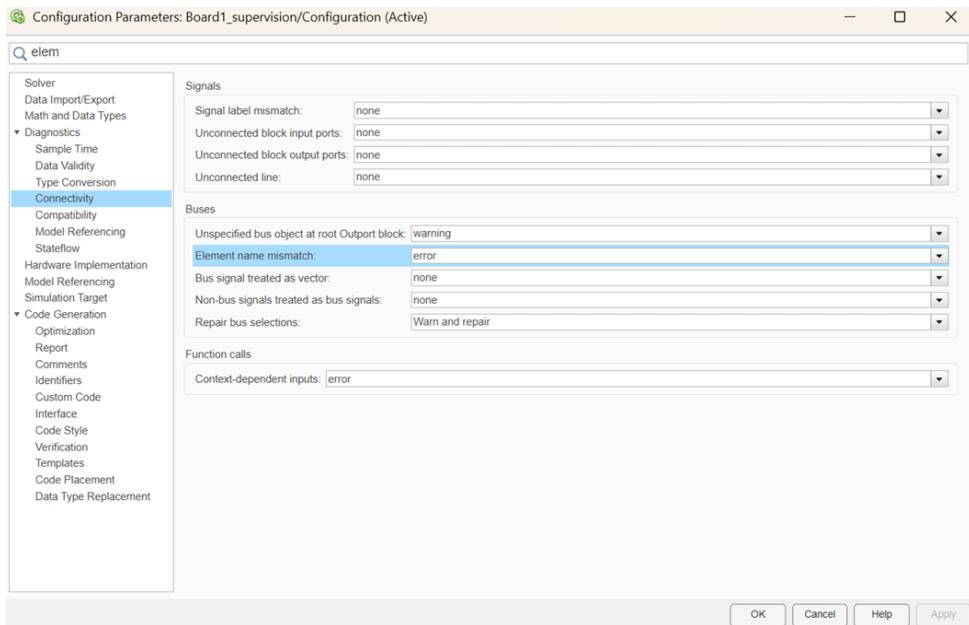


Figura 7.3: Impostazione di element name mismatch

### 7.2.3 Definizione delle enumerazioni

Un ulteriore requisito per la generazione del codice è la definizione delle enumerazioni utilizzate per gestire gli stati logici dell'applicazione. Tali definizioni, salvate come classi MATLAB (.m), devono essere rese accessibili all'interno dell'ambiente di sviluppo. Pertanto, è necessario configurare l'ambiente aggiungendo la directory contenente le definizioni delle enumerazioni al MATLAB Search Path. Questa operazione garantisce che Simulink possa risolvere correttamente i riferimenti ai tipi enumerativi durante la fase di compilazione e linking, evitando errori di "simbolo non trovato".

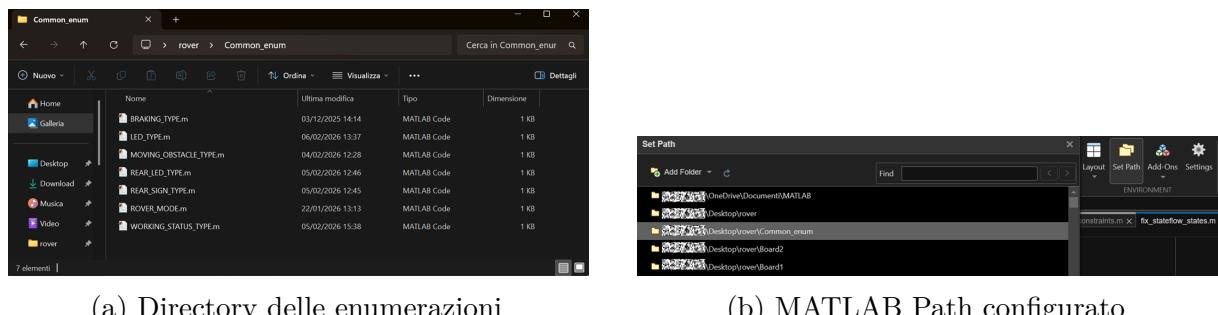


Figura 7.4: Integrazione delle enumerazioni nel progetto: a sinistra la struttura della cartella contenente i file di enumerazione; a destra la configurazione del MATLAB Path con l'aggiunta della directory.

## 7.3 Configurazione del Solver e criticità temporali

Per la generazione del codice è stato configurato un **Fixed-step Discrete Solver**.

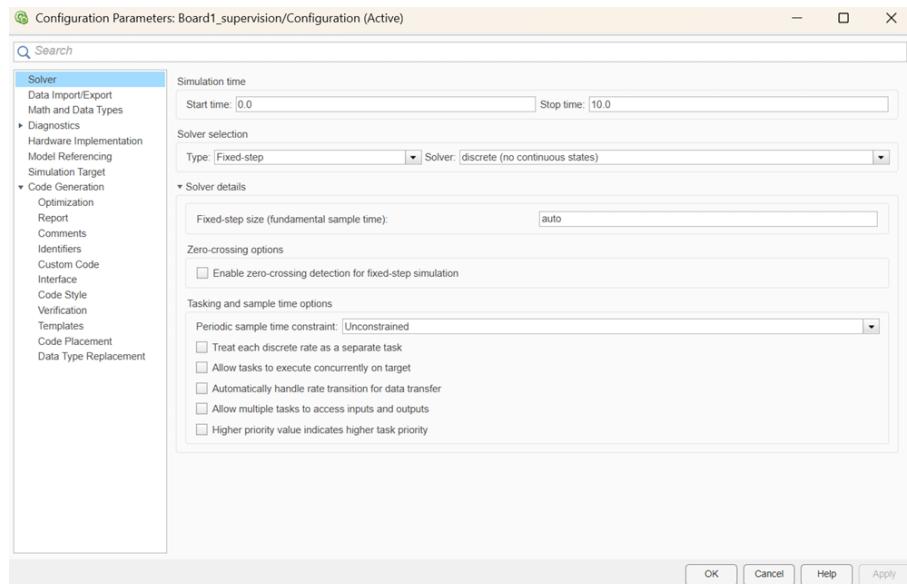


Figura 7.5: Scelta del Fixed Step Solver

Poiché il firmware viene eseguito su microcontrollore con FreeRTOS, il codice generato assume un tempo di campionamento costante  $T_s$ .

Le primitive temporali di Stateflow (ad esempio `after()`) non misurano il tempo reale, ma il numero di chiamate alla funzione `Board_step()`.

Nel design adottato, la funzione `Board_step()` non viene richiamata una sola volta per ciclo di task, ma deve essere invocata iterativamente in un ciclo `do-while` finché l'automa non raggiunge uno stato finale.

```

1 void executeSupervision(void){
2
3     do{
4         Board1_step();
5     }
6     while((Board1_DW.is_Supervisor != Board1_IN_Same_decision) &&
7           (Board1_DW.is_Single_Board != Board1_IN_Other_board_failure) &&
8           (Board1_DW.is_Degraded != Board1_IN_Restarting) &&
9           (Board1_DW.is_Restablish != Board1_IN_Connection_reestablished));
10 }
```

Il problema si manifesta nell'intervallo tra due attivazioni successive della task FreeRTOS. Durante tale intervallo il modello è inattivo, ma il codice generato non ha percezione del tempo reale trascorso. Alla successiva chiamata, l'esecuzione riprende come se fosse trascorso un solo istante, generando una discrepanza tra tempo reale e tempo simulato interno.

## 7.4 Strategia di disaccoppiamento e Stub Functions

Per garantire la portabilità del codice e risolvere le criticità legate al passo temporale, è stata adottata una strategia di **disaccoppiamento totale**.

Le primitive temporali native di Stateflow (come `after`) sono state rimosse e sostituite da chiamate a funzioni esterne (Stub Functions). Lo stesso approccio è stato esteso alle interazioni con l'hardware e ai protocolli di comunicazione.

Poiché MATLAB non dispone delle librerie hardware specifiche (HAL STM32, FreeRTOS), le funzioni di interfaccia, contenute nei file `board*_functions.c`, sono inizialmente implementate come funzioni vuote (dummy). Ciò consente al generatore di codice di completare la compilazione senza errori di linking.

Successivamente, in fase di integrazione nell'IDE embedded, il corpo di tali funzioni viene completato con le chiamate reali ai driver e al sistema operativo.

Affinché il processo abbia esito positivo, è necessario includere la directory contenente questi file nel MATLAB Search Path prima di avviare la generazione.

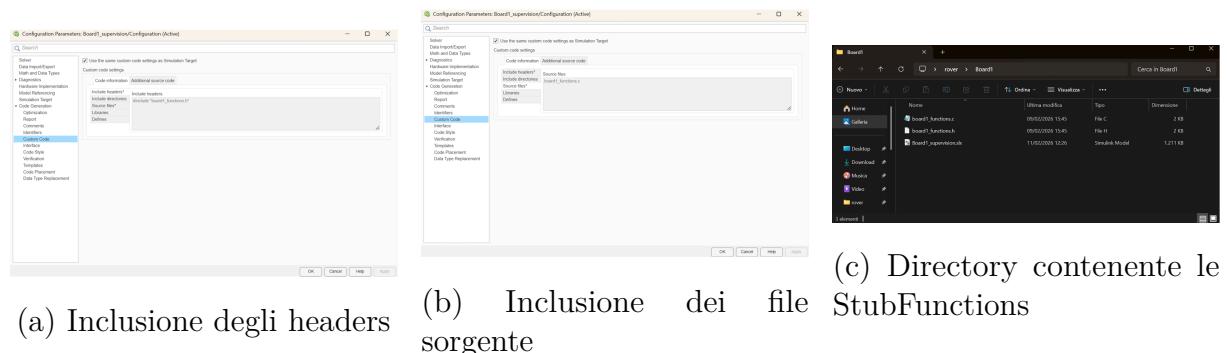


Figura 7.6: Stub functions generate automaticamente da Embedded Coder

## 7.5 Script di post-processing

L'impostazione conclusiva, essenziale per l'integrazione del sistema, riguarda la configurazione di uno script di post-processing all'interno dei parametri di generazione del codice. Una volta terminata la creazione dei file sorgente, Embedded Coder invoca automaticamente la routine

```
fix_stateflow_states.m
```

Questo strumento è stato sviluppato per colmare il divario strutturale tra l'output standard di Simulink e l'architettura specifica richiesta dal progetto firmware finale.

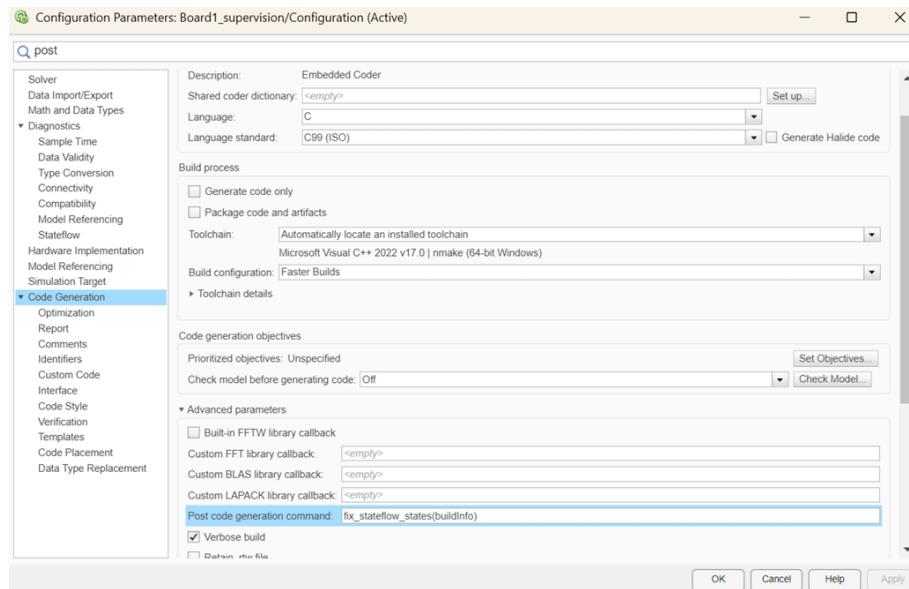


Figura 7.7: Configurazione dello script di post-processing

### 7.5.1 Esposizione degli stati

Per impostazione predefinita, Embedded Coder ottimizza il codice C incapsulando le definizioni degli stati interni direttamente all'interno dei file sorgente .c. Sebbene questa scelta sia coerente con principi di efficienza e incapsulamento, essa comporta che tali definizioni rimangano di fatto “private”, risultando non accessibili agli altri moduli software del firmware. In un sistema strutturato su più componenti cooperanti, questa limitazione può ostacolare la possibilità di osservare o utilizzare lo stato corrente della macchina a stati al di fuori del modulo che la implementa.

L'analisi automatica dei file generati consente di individuare tali definizioni e di trasferirle dal file sorgente locale (.c) al corrispondente file header (.h). In questo modo, le costanti che rappresentano gli stati diventano parte dell'interfaccia pubblica del modulo. Tale modifica permette agli altri componenti del firmware di interrogare in maniera strutturata lo stato corrente della macchina a stati, agevolando le attività di monitoraggio, diagnostica e integrazione con le restanti parti del sistema.

### 7.5.2 Integrazione automatica nel repository

Lo script gestisce inoltre l'integrazione automatica del codice generato nel repository firmware.

In particolare:

- Analizza il nome del modello per determinare la scheda di destinazione.
- Copia automaticamente i file generati:
  - Board\*.c
  - Board\*.h
  - Board\*\_private.h
  - Board\*\_types.h
  - rtwtypes.h

- Inserisce i file nelle directory Core/Src e Core/Inc.

### NOTA

I percorsi di destinazione sono configurabili all'interno dello script.

Questo meccanismo garantisce che, ad ogni nuova generazione, il progetto firmware venga automaticamente aggiornato, eliminando la necessità di copia manuale dei file.

## 7.6 Procedura operativa di generazione

La generazione del codice viene eseguita direttamente sul componente Stateflow di interesse:

1. Selezionare il blocco Chart all'interno del modello.
2. Cliccare con il tasto destro.
3. Selezionare **C/C++ Code > Build This Subsystem**.

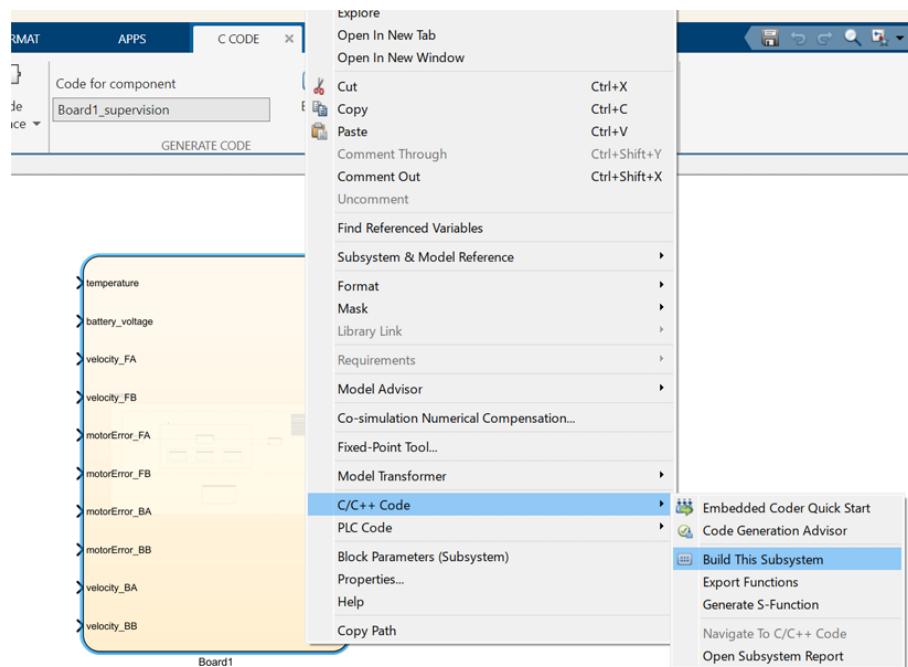


Figura 7.8: Enter Caption

Il completamento della procedura senza errori conferma:

- la corretta traduzione del diagramma a stati in codice C,
- l'esecuzione dello script di post-processing,
- l'aggiornamento automatico del progetto firmware.

# **Parte III**

# **Sistema Operativo Real Time**

---

# Schedulazione Real Time

---

Questo capitolo descrive la progettazione della schedulazione real-time dei task software eseguiti sulle due Board STM32G4 del rover, con l'obiettivo di garantire il rispetto dei vincoli temporali imposti dal controllo dei motori, dalla percezione degli ostacoli e dalle funzioni di sicurezza. La progettazione è condotta assumendo un sistema operativo real-time preemptive a priorità fissa, nel quale l'intero comportamento del sistema è modellato come un insieme di task periodici, ciascuno caratterizzato da periodo, priorità e worst-case execution time. La schedulazione è progettata separatamente per le due Board, in quanto esse svolgono ruoli funzionali distinti.

- La **Board 1** è dedicata alle funzioni di controllo in retroazione dei motori e al monitoraggio dei parametri critici di funzionamento (encoder, batteria, temperatura).
- La **Board 2** è invece responsabile della percezione dell'ambiente e della ricezione dei comandi remoti

Nei paragrafi successivi vengono quindi identificati i task eseguiti su ciascuna Board, stimati i rispettivi tempi di esecuzione nel caso peggiore e motivate le scelte di periodo, priorità e interazione tra task, al fine di dimostrare che il sistema soddisfa i requisiti real-time imposti dalle specifiche di progetto.

## 8.1 Spazio e tempo di frenata

La determinazione dello spazio e del tempo di frenata del rover richiede, come primo passo, la definizione della velocità massima di avanzamento raggiungibile dal sistema in condizioni nominali. Tale valore costituisce infatti il caso peggiore dal punto di vista della sicurezza e viene assunto come riferimento per la progettazione dei meccanismi di arresto di emergenza e per la definizione delle **deadline hard real-time** associate alle funzioni di percezione degli ostacoli.

I motori utilizzati presentano le seguenti caratteristiche, fornite dal datasheet del costruttore:

- velocità a vuoto: 170 RPM ( $\pm 10\%$ );
- velocità a carico: 145 RPM ( $\pm 10\%$ );

La presenza di una tolleranza del  $\pm 10\%$  sulla velocità nominale implica che motori diversi possano presentare velocità di rotazione differenti a parità di comando. Al fine di garantire un comportamento coerente e simmetrico del rover si è scelto di limitare via software la velocità massima di rotazione a un valore conservativo, sicuramente raggiungibile da tutti i motori anche nel caso peggiore.

Per la determinazione della velocità massima di riferimento si è scelto deliberatamente di assumere come base di calcolo la *velocità a vuoto* del motore, nonostante essa non rappresenti una condizione di funzionamento realistica durante il moto del rover. Tale scelta è stata effettuata in modo intenzionale al fine di collocare l'analisi in uno scenario *pessimistico* dal punto di vista della sicurezza, sovrastimando la velocità massima potenzialmente raggiungibile dal sistema.

Considerando la tolleranza del  $\pm 10\%$  sulla velocità a vuoto, il limite inferiore risulta pari

a:

$$n_{\max} = 153 \text{ RPM},$$

successivamente tale valore è stato limitato via software a:

$$n_{\max} = 150 \text{ RPM},$$

Il rover è equipaggiato con ruote di diametro pari a:

$$D = 158 \text{ mm} = 0.158 \text{ m.}$$

La circonferenza della ruota risulta pertanto:

$$C = \pi D \approx 0.496 \text{ m.}$$

Convertendo la velocità di rotazione massima in giri al secondo e considerando la circonferenza della ruota, si ottiene:

$$v_{\max} = \pi D \cdot \frac{n_{\max}}{60} \approx 1.24 \text{ m/s.}$$

A partire da tale velocità, è possibile determinare il margine temporale disponibile per l'arresto del rover in presenza di un ostacolo rilevato lungo la direzione di marcia.

Dalle specifiche di progetto del sistema risulta che il rover deve essere in grado di rilevare e gestire un ostacolo improvviso quando questo si trovi a una distanza inferiore a 70 cm rispetto alla parte anteriore del veicolo. Tale valore rappresenta la soglia nominale di sicurezza definita a livello di specifica. Al fine di collocare l'analisi in una condizione ulteriormente prudenziale, si è scelto deliberatamente di considerare una distanza di rilevamento significativamente inferiore rispetto a quella prescritta, pari a:

$$d = 0.30 \text{ m.}$$

Questa scelta consente di ridurre ulteriormente lo spazio disponibile per la frenata, ponendo il sistema in uno scenario più gravoso rispetto a quello previsto dalle specifiche e garantendo un margine di sicurezza aggiuntivo nella verifica dei vincoli temporali.

Si considera pertanto il caso peggiore in cui il rover proceda alla velocità massima  $v_{\max}$  e un ostacolo venga rilevato esattamente alla distanza  $d = 0.30 \text{ m}$ . In tale condizione, l'intero intervallo spaziale disponibile deve essere utilizzato per ridurre la velocità del rover fino all'arresto completo, senza che avvenga il contatto con l'ostacolo.

Assumendo, in prima approssimazione, un moto rettilineo uniforme fino all'istante di arresto, il tempo massimo teorico a disposizione per la frenata può essere calcolato come:

$$t_{\max} = \frac{d}{v_{\max}} = \frac{0.30}{1.24} \approx 0.24 \text{ s.}$$

Il valore  $t_{\max} \approx 240 \text{ ms}$  rappresenta quindi il tempo complessivo entro il quale il sistema deve:

- acquisire la misura di distanza dai sensori a ultrasuoni;
- riconoscere la condizione di pericolo a livello di supervisione;
- generare il comando di arresto;
- applicare l'azione di frenata fino all'azzeramento della velocità.

Alla luce delle stime precedenti, nei paragrafi successivi tale vincolo verrà formalizzato introducendo una *deadline* di progetto pari a **220 ms**, utilizzata come riferimento per l'analisi temporale e la verifica di schedulabilità dei task.

## 8.2 Identificazione del task set

Il task set del rover è stato strutturato per rispondere ai requisiti di un sistema distribuito, richiedendo l'esecuzione coordinata su processori differenti. L'utilizzo di Simulink e della generazione automatica del codice ha dettato ulteriori specifici vincoli implementativi.

La generazione automatica sviluppata da Stateflow fornisce funzioni di step che presuppongono un rate di esecuzione deterministico, poco adatto alla nostra necessità di ottenere una funzione in grado di eseguire l'intero ciclo di supervisione distribuita. Poiché la modifica manuale del codice generato invaliderebbe le certificazioni di sicurezza, è stato necessario intervenire a livello modellistico.

Lo schema Simulink, come già illustrato precedentemente, è stato quindi ingegnerizzato per essere integrato in un ciclo iterativo gestito da una wrapper function esterna: `executeSupervision()`.

Dal punto di vista dello scheduling, questa funzione rappresenta l'unità atomica di supervisione: ad ogni invocazione, essa elabora gli input e calcola gli output per il sistema, coprendo sia lo stato di funzionamento normale che quello degradato. La separazione del modello in due automi Stateflow distinti ha infine permesso di generare e distribuire il codice specifico per ciascuna delle due schede di controllo.

Alla luce di tali premesse, la scelta progettuale definitiva è ricaduta sull'implementazione della logica di supervisione come un task autonomo (standalone). Tale task, presente su entrambe le schede, è incaricato di elaborare lo stato corrente del sistema al fine di generare i nuovi riferimenti di attuazione.

A supporto del task di supervisione, l'architettura finale prevede specifici task di I/O, definiti per ciascuna board in funzione delle periferiche connesse e dei requisiti temporali associati alla loro gestione.

**Task Set per la Board 2 (Master)** Nel caso della **Board Master**, l'organizzazione delle operazioni di I/O non prevede una separazione rigida in task periodici distinti per ciascuna fase funzionale. In fase di progettazione è stata valutata la possibilità di modellare separatamente l'acquisizione dei dati via bus I<sup>2</sup>C (ricevitore del radiocomando e unità inerziale MPU6050) e l'attuazione in anello aperto; tuttavia, tale suddivisione non risulta giustificata dal comportamento temporale del sistema.

Le fasi di lettura degli ingressi, elaborazione della logica di supervisione e applicazione dei comandi sono infatti strettamente accoppiate e devono essere eseguite **in modo sequenziale** all'interno dello stesso ciclo decisionale, al fine di garantire la coerenza temporale tra dati acquisiti e comandi applicati. La loro attivazione non è quindi indipendente, ma avviene sempre secondo un ordine deterministico.

La separazione in task distinti avrebbe richiesto l'introduzione di artifizi per ricostruire una sequenzialità già imposta a livello logico, senza introdurre parallelismo reale e con un aumento dell'overhead di scheduling. Per queste ragioni, le operazioni di acquisizione via I<sup>2</sup>C e di attuazione sono state integrate direttamente nel task di supervisione della Board Master, modellando l'intero ciclo come un'unica transazione temporale e migliorando la prevedibilità temporale end-to-end del sistema.

La Board 2 prevede un ulteriore compito dedicato alla gestione della telemetria, distinto dal ciclo di supervisione. Tale funzionalità consiste nella raccolta, aggregazione e trasmissione verso l'operatore delle principali informazioni di stato del sistema. La telemetria è stata progettata come task separato poiché non condivide i medesimi vincoli temporali del ciclo di controllo: mentre la supervisione deve rispettare una cadenza stretta e deterministica per

garantire stabilità e coerenza decisionale, la trasmissione delle informazioni può avvenire con frequenza inferiore senza influenzare le prestazioni dinamiche del sistema.

Di conseguenza, la Board Master possiede, oltre al già citato task di supervisione, i seguenti task dedicati:

- **readSonarTask**: Responsabile dell'acquisizione dati dai sonar frontali
- **telemetryLoggerTask**: dedicato alla costruzione e trasmissione del pacchetto di telemetria verso il controller.

**Task Set per la Board 1 (Slave)** Per quanto concerne la **Board Slave**, il ragionamento di accorpamento dei task periodici all'interno del task di supervisione non è stato applicato. Tale scelta è motivata dal fatto che l'integrazione delle operazioni di lettura nel ciclo di supervisione avrebbe introdotto una latenza aggiuntiva nel momento in cui la Board Master richiede dati alla Slave durante la comunicazione inter-board, latenza dovuta all'eventuale esecuzione precedente di una fase di acquisizione. Poiché i dati acquisiti dalla Board Slave non presentano un livello di criticità tale da richiederne la lettura on-demand con la richiesta del Master, si è ritenuto più opportuno demandare tali operazioni a task autonomi. Sono pertanto presenti i seguenti task:

- **readSensorTask**: Responsabile del monitoraggio della telemetria, inclusa la tensione della batteria, la temperatura della CPU e la velocità media dei motori.
- **pidTask**: Dedicato all'esecuzione dell'algoritmo di controllo in anello chiuso (PID) per i motori, configurato con un periodo di esecuzione rigoroso di 5 ms.
- **lightTask**: Incaricato della gestione dell'illuminazione del rover.

Questa configurazione si è rivelata determinante per garantire le prestazioni del sistema distribuito. In particolare, per la Board Slave, l'architettura RTOS permette di rispettare i vincoli temporali stringenti del controllo PID (5 ms). Parallelamente, l'adozione della medesima struttura a task sulla Board Master, ha consentito una gestione efficiente della lettura dei sonar, sensori che necessitano di molto tempo per la lettura e che se fossero stati gestiti in maniera sequenziale all'interno di un unico task di lettura input avrebbero rallentato notevolmente la supervisione di sistema e la reattività ai comandi.

## 8.3 Sincronizzazione e dipendenze tra task

Come la maggior parte dei sistemi di controllo, l'architettura di supervisione del rover è modellabile logicamente come un ciclo sequenziale costituito da tre fasi:

1. acquisizione degli ingressi;
2. elaborazione dati;
3. aggiornamento degli attuatori.

Tale dinamica è intrinsecamente sequenziale, in quanto vincolata dalla causalità: non è possibile elaborare dati non ancora acquisiti né attuare comandi non ancora calcolati. In prima analisi, l'adozione di un RTOS (Real-Time Operating System) potrebbe apparire controproducente, introducendo un overhead di sistema apparentemente ingiustificato rispetto a una logica bare-metal sequenziale. Tuttavia, l'architettura distribuita del rover impone requisiti temporali eterogenei che rendono l'RTOS una scelta imprescindibile.

Esaminando i task funzionali, emerge chiaramente come alcune periferiche richiedano frequenze di aggiornamento disaccoppiate dal ciclo principale di supervisione. Per quanto riguarda la scheda Slave, il controllo in retroazione (PID) necessita di una frequenza operativa di **200 Hz**, decisamente superiore a quella del ciclo di supervisione. Analogamente, il sottosistema di illuminazione richiede un aggiornamento costante per gestire le animazioni dell'**array LED** posteriore in modo fluido e indipendente dal carico della CPU.

Anche la scheda Master, pur sembrando meno soggetta a vincoli di parallelismo, trae un significativo vantaggio dall'RTOS. Parallelizzando l'acquisizione dei sensori a ultrasuoni rispetto al ciclo di supervisione, è possibile rendere il sistema più reattivo: i dati sensoriali vengono acquisiti in background e resi disponibili per l'iterazione successiva. Questa strategia, pur introducendo una concorrenza controllata, rispetta rigorosamente i vincoli hard real-time e garantisce una responsività ai comandi utente altrimenti irraggiungibile. Tuttavia, l'introduzione del multitasking e della parallelizzazione non è esente da criticità: l'accesso concorrente alle risorse condivise da parte di task a priorità diversa introduce infatti il rischio di race conditions e incoerenza dei dati.

### Esempio critico

Un esempio critico di tale problematica si riscontra nel controllo PID: qualora il task di esecuzione dell'algoritmo tentasse di prelevare i valori target (setpoint) nell'istante esatto in cui il ciclo di supervisione li sta aggiornando, il sistema acquisirebbe dati incoerenti. Tale condizione comprometterebbe il funzionamento del controllo, causando instabilità nel moto e potenziali situazioni di pericolo.

Al fine di mitigare tali criticità, la strategia di gestione delle risorse condivise ha privilegiato l'impiego di **sezioni critiche** rispetto all'adozione di **semafori a mutua esclusione (mutex)**. Tale scelta architettonica è motivata dalla natura estremamente breve delle operazioni da proteggere, il cui worst-case execution time risulta pari a circa **940 ns**. In questo contesto, l'utilizzo di mutex avrebbe introdotto un overhead di gestione sproporzionato rispetto al tempo di esecuzione effettivo delle operazioni, risultando inefficiente dal punto di vista temporale.

Analizzando la schedulazione sulla **Board Master**, emerge che, nonostante la presenza di due task concorrenti, non si verifica una reale sovrapposizione temporale nell'accesso alle strutture dati condivise. I task operano infatti secondo il principio di singola responsabilità e con periodicità fissa all'interno del macro-ciclo di supervisione; di conseguenza, la coerenza dei dati sarebbe, in linea teorica, garantita dalla sequenzialità intrinseca delle operazioni. Ciononostante, in un'ottica di **programmazione difensiva**, le primitive di sincronizzazione sono state comunque implementate anche sulla Board Master. Questa scelta mira esclusivamente a rendere il sistema robusto rispetto a potenziali errori di logica o a future modifiche del codice, prevenendo a priori l'insorgenza di condizioni di gara dovute a variazioni involontarie del flusso di esecuzione.

Per quanto concerne la **Board Slave**, il contesto di esecuzione risulta più critico, poiché la totalità dei task opera effettivamente in concorrenza, esponendo il sistema a un rischio concreto di race condition. In tale scenario, l'adozione delle sezioni critiche si è rivelata la soluzione più efficiente: data la brevissima durata delle operazioni protette, questo approccio introduce un jitter trascurabile (inferiore a 1  $\mu$ s), garantendo l'integrità dei dati senza penalizzare le prestazioni complessive del sistema.

## 8.4 Dimensionamento dei task

Questa sezione è dedicata al dimensionamento temporale dei task che compongono il sistema real-time del rover. In particolare, per ciascun task vengono analizzati il Worst Case Execution Time (WCET) e il periodo di attivazione, al fine di verificare il rispetto dei vincoli temporali imposti dalle specifiche di progetto. Il dimensionamento è condotto combinando stime analitiche e misure sperimentali, adottando in modo sistematico ipotesi conservative, così da garantire che i parametri scelti costituiscano una base solida per la successiva analisi di schedulabilità del sistema.

### 8.4.1 Analisi generale del supervisionTask

Il Task di Supervisione (`supervisionTask`) costituisce il centro nevralgico dell'intera architettura firmware. Il suo ruolo fondamentale è quello di orchestrare il comportamento autonomo del rover, eseguendo gli algoritmi decisionali che trasformano le informazioni grezze provenienti dai sensori in comandi operativi per la navigazione e la sicurezza.

Ai fini dell'analisi del WCET (Worst-Case Execution Time), trattandosi di un sistema di controllo distribuito, è cruciale determinare con precisione le latenze introdotte dal protocollo di comunicazione. La trasmissione dei dati rappresenta, infatti, un vincolo temporale rigido per la sincronizzazione tra le board.

Considerando l'utilizzo di una periferica UART gestita in DMA con un baud rate di 115200 bps e un frame che prevede, per ogni byte di dati (8 bit), l'aggiunta di 1 bit di start e 1 bit di stop, il tempo di trasmissione  $T_{tx}$  è definito dalla relazione:

$$T_{tx} = \frac{N_{byte} \times 10}{115200}$$

Di seguito si riporta il calcolo dettagliato per ogni tipologia di pacchetto scambiato:

- Stato Locale Board 1 (Slave): La struttura ha una dimensione di 20 byte. Includendo i 4 byte del CRC-32, il payload totale ammonta a 24 byte.

$$T_{B1} = \frac{(20 + 4) \times 10}{115200} \approx 2.1 \text{ ms}$$

- Stato Locale Board 2 (Master): La struttura ha una dimensione di 23 byte. Per garantire l'allineamento in memoria (multiplo di 4 byte), viene aggiunto 1 byte di padding, portando la base a 24 byte. Inclusi i 4 byte del CRC-32, il totale da trasmettere è di 28 byte.

$$T_{B2} = \frac{(23 + 4) \times 10}{115200} \approx 2.43 \text{ ms}$$

- Stato Globale: La struttura aggregata ha una dimensione effettiva di 49 byte. Per garantire l'allineamento in memoria (multiplo di 4 byte), vengono aggiunti 3 byte di padding, portando la base a 52 byte. Sommando i 4 byte di CRC, il pacchetto finale risulta di 56 byte.

$$T_{Global} = \frac{(49 + 3_{pad} + 4) \times 10}{115200} \approx 4.9 \text{ ms}$$

- Decisione: La struttura decisionale occupa 24 byte. Con l'aggiunta dei 4 byte di CRC, il totale trasmesso è di 28 byte.

$$T_{Dec} = \frac{(24 + 4) \times 10}{115200} \approx 2.43 \text{ ms}$$

I parametri temporali che regolano le transizioni di timeout all'interno della macchina a stati (Stateflow) sono strettamente correlati ai tempi di trasmissione calcolati nell'analisi del WCET.

Per garantire la robustezza del protocollo e prevenire falsi positivi dovuti a lievi jitter o latenze intrinseche nella gestione del DMA, non si utilizzano i tempi teorici puri. È stata invece adottata una convenzione che applica un margine di sicurezza secondo la seguente logica:

1. Timeout di Invio: Calcolato sommando al tempo teorico di trasmissione ( $T_{tx}$ ) un margine di 0.5 ms. Questo assorbe le latenze di configurazione del DMA e l'overhead del software.
2. Timeout di Ricezione: Calcolato sommando al timeout di invio della controparte un ulteriore margine di 0.5 ms, per compensare eventuali disallineamenti nello start della ricezione.

Di seguito si riportano i valori definitivi configurati nel firmware:

**Scambio Stato Locale** A causa della diversa dimensione delle strutture dati tra Master e Slave, i timeout sono asimmetrici:

- Board 1 (Slave):
  - Tempo massimo per l'invio del proprio stato: 2.6 ms (STATE\_SEND\_TIMEOUT)
  - Tempo massimo di attesa per ricevere lo stato della Board 2: 3.5 ms (STATE\_RECEIVE\_TIMEOUT)
- Board 2 (Master):
  - Tempo massimo per l'invio del proprio stato: 3.0 ms (STATE\_SEND\_TIMEOUT)
  - Tempo massimo di attesa per ricevere lo stato della Board 1: 3.1 ms (STATE\_RECEIVE\_TIMEOUT)

**Scambio Stato Globale** In questa fase le dimensioni dei pacchetti sono consistenti, portando a valori unificati:

- Tempo massimo di invio: 5.4 ms (GLOBAL\_STATE\_SEND\_TIMEOUT)
- Tempo massimo di ricezione: 5.9 ms (GLOBAL\_STATE\_RECEIVE\_TIMEOUT)

**Scambio della Decisione** Analogamente, per la struttura decisionale:

- Tempo massimo di invio: 3.0 ms (DECISION\_SEND\_TIMEOUT)
- Tempo massimo di ricezione: 3.5 ms (DECISION\_RECEIVE\_TIMEOUT)

**Segnalazione e Handshake Hardware** Per le operazioni di controllo a basso livello e la sincronizzazione iniziale:

- Trasmissione Single Byte: Utilizzato nei ping.
  - Invio: 0.5 ms (BYTE\_SEND\_TIMEOUT)
  - Ricezione: 1.0 ms (BYTE\_RECEIVE\_TIMEOUT)
- Sincronizzazione Linee: Tempo massimo di attesa per la transizione logica delle linee fisiche (Talk): 0.5 ms (WAIT\_TIMEOUT).
- Startup Iniziale: Tempo concesso al Master (Board 2) all'avvio della sessione per attendere la prontezza della controparte: 1.5 ms (INITIAL\_TIMEOUT).

Per determinare il WCET del task di supervisione, è necessario identificare il percorso di esecuzione più lungo possibile all'interno della macchina a stati. A livello implementativo, il task gestisce l'evoluzione del protocollo tramite un ciclo do-while, che itera l'esecuzione degli step della macchina a stati finché non viene raggiunta una condizione terminale. Le condizioni di uscita dal ciclo, che determinano la fine del task per il periodo corrente, sono le seguenti:

1. Consenso Raggiunto: Le board completano lo scambio, le decisioni coincidono e viene validata l'attuazione.
2. Transizione in Degradato: Si verificano errori di comunicazione persistenti (timeout/CRC) che portano allo stato Degraded.
3. Fallimento del Ripristino: Partendo da uno stato degradato, il tentativo di handshake (Ping) fallisce, confermando la modalità degradata.
4. Divergenza sullo Stato Globale: Le board comunicano correttamente ma rilevano stati globali diversi, forzando la modalità Single Board.
5. Divergenza sulla Decisione: Il ciclo si completa ma le decisioni finali non coincidono, forzando la modalità Single Board.

Il caso peggiore non corrisponde a un fallimento immediato (che interromperebbe il ciclo prematuramente), ma allo scenario in cui il protocollo viene eseguito nella sua interezza, arrivando fino al confronto finale delle decisioni (indipendentemente dall'esito positivo o negativo di tale confronto), subendo però il massimo numero di ritardi ammissibili.

Considerando che il protocollo tollera al massimo una ritrasmissione per fase prima di abortire, il Worst Case si verifica quando il sistema è costretto a ritrasmettere i pacchetti più onerosi in termini di tempo, ovvero quelli dello Stato Globale, sia in direzione Master → Slave che Slave → Master.

Di seguito il calcolo dettagliato per le due unità, ottenuto sommando i timeout definiti nella sezione precedente: Il Master deve attendere il timeout iniziale di wakeup e gestire le attese per le ritrasmissioni dello stato globale. La formula risultante è:

$$\begin{aligned} WCET_{B2} = & T_{Initial} + T_{StateTX} + T_{StateRX} \\ & + 2 \cdot (T_{GlobalTX} + T_{GlobalRX}) \\ & + T_{DecTX} + T_{DecRX} \\ & + 6 \cdot T_{Wait} \end{aligned}$$

Sostituendo i valori numerici:

$$1.5 + 3.0 + 3.1 + 2 \cdot (5.4 + 5.9) + 3.0 + 3.5 + 6 \cdot (0.5) = \mathbf{39.7 \text{ ms}}$$

**Board 1 (Slave)** Lo Slave non contempla il timeout iniziale, ma ha un diverso numero di interazioni di attesa sulle linee di controllo (Wait Timeout):

$$\begin{aligned}WCET_{B1} = & T_{StateTX} + T_{StateRX} \\& + 2 \cdot (T_{GlobalTX} + T_{GlobalRX}) \\& + T_{DecTX} + T_{DecRX} \\& + 7 \cdot T_{Wait}\end{aligned}$$

Sostituendo i valori numerici:

$$2.6 + 3.5 + 2 \cdot (5.4 + 5.9) + 3.0 + 3.5 + 7 \cdot (0.5) = \mathbf{38.7 \text{ ms}}$$

La natura asimmetrica del protocollo di comunicazione impone strategie di schedulazione differenziate per le due unità di elaborazione.

### 8.4.2 Board 2

#### supervisionTask (Board 2)

Come discusso nelle sezioni precedenti, sulla **Board Master** le operazioni di acquisizione degli ingressi, di elaborazione della logica di supervisione e di attuazione dei comandi sono state intenzionalmente accorpate all'interno di un unico ciclo decisionale. Di conseguenza, le operazioni di lettura dei dispositivi connessi al bus I<sup>2</sup>C non sono implementate come task periodico autonomo, ma vengono eseguite direttamente all'interno della *supervisionTask*, nella fase di acquisizione degli ingressi del ciclo di supervisione.

In particolare, ad ogni attivazione della *supervisionTask* vengono acquisiti:

- i comandi provenienti dal ricevitore del controller remoto;
- i dati inerziali (accelerometro e giroscopio) forniti dal sensore MPU6050.

**Ricevitore controller** La comunicazione dei comandi di controllo avviene tramite un ricevitore basato su microcontrollore ESP32-S3, che riceve i dati da un nodo trasmittente *ESP32* mediante protocollo *ESP-NOW*.

Dal punto di vista della Board STM32, il ricevitore ESP32-S3 è visto come una periferica *slave* I<sup>2</sup>C sincrona, interrogata periodicamente per l'acquisizione dello stato corrente del controller remoto.

Il bus I<sup>2</sup>C è configurato in *Standard Mode* e pertanto ha una frequenza di clock pari a  $f_{I^2C} = 100$  kHz. Nel protocollo I<sup>2</sup>C, ciascun byte trasferito è codificato su 8 bit di dato seguiti da un bit di ACK/NACK. Pertanto, il trasferimento di un singolo byte richiede 9 bit. Il tempo necessario alla trasmissione dei dati utili può essere calcolato come:

$$T_{data} = \frac{N_{data} \cdot 9bit}{f_{I^2C}}.$$

Il ricevitore ESP32-S3 rende disponibile alla STM32 la seguente struttura dati:

```

1 typedef struct __attribute__((packed)) {
2     uint16_t ax;
3     uint16_t ay;
4     uint8_t a_btn;
5
6     uint16_t bx;
7     uint16_t by;
8     uint8_t b_btn;
9
10    uint8_t btn1;
11    uint8_t btn2;
12    uint8_t btn3;
13    uint8_t btn4;
14 } controller_data_t;
15
16 typedef struct __attribute__((packed)) {
17     uint8_t alive;
18     uint8_t controller_percentage;
19     controller_data_t controller_data;
20 } controller_information_t;

```

Listing 8.1: Struct packed per dati controller

La dimensione complessiva del payload risulta:

$$\begin{aligned} N_{bit,data} &= |\text{controller\_information\_t}| = \\ &|\text{alive}| + |\text{controller\_percentage}| + |\text{controller\_data\_t}| \\ &= (1 + 1 + 14) \text{ byte} = 16 \text{ byte}. \end{aligned}$$

Sostituendo il valore del payload trasferito, pari a  $N_{data} = 16$  byte, si ottiene:

$$T_{data} = \frac{16 \cdot 9}{100\,000} \text{ s} = 1.44 \text{ ms}.$$

Oltre al payload, la comunicazione I<sup>2</sup>C prevede la trasmissione dell'indirizzo dello slave (con bit aggiuntivo di read-write) e dei bit di START e STOP. Nel caso di indirizzamento a 7 bit, l'indirizzo viene inviato insieme al bit di read-write, seguito da un bit di ACK, per un totale di 9 bit aggiuntivi. Il tempo associato alla trasmissione dell'indirizzo risulta quindi:

$$T_{addr} = \frac{9}{f_{I^2C}} = \frac{9}{100\,000} \text{ s} = 90 \mu\text{s}.$$

Le condizioni di *START* e *STOP* non trasportano dati, ma occupano comunque il bus per un intervallo temporale non nullo: altri 2 bit. Ne consegue:

$$T_{\text{START+STOP}} = \frac{2}{f_{I^2C}} \text{ s} = 20 \mu\text{s}.$$

La latenza totale necessaria alla lettura della struttura `controller_information_t` dal ricevitore ESP32-S3 può quindi essere espressa come:

$$C_{I^2C,RX} = T_{addr} + T_{data} + T_{\text{START+STOP}} = 90 \mu\text{s} + 1.44 \text{ ms} + 20 \mu\text{s} = 1.55 \text{ ms}.$$

Tale tempo può essere assunto come Worst Case Execution Time della porzione del task `supervisionTask` relativa alla lettura del ricevitore ESP32-S3, in quanto in questa transazione non vi è variabilità né nella dimensione del payload trasferito né nel flusso di esecuzione.

**Accelerometro/Giroscopio MPU6050** Il sensore MPU6050 integra un accelerometro triassiale e un giroscopio triassiale ed è connesso alla Board Master tramite bus I<sup>2</sup>C.

La lettura dei registri del MPU6050 avviene tramite una transazione I<sup>2</sup>C di tipo *memory read*, strutturata nelle seguenti due fasi consecutive:

1. **Fase di selezione del registro interno (write):** il master genera la condizione di *START* (1 bit), trasmette l'indirizzo a 7 bit dello slave con bit di scrittura e relativo ACK (9 bit complessivi) e invia un byte contenente l'indirizzo del registro interno da cui avviare la lettura (GYRO\_XOUT\_H o ACCEL\_XOUT\_H, 9 bit complessivi).
2. **Fase di lettura dei dati (read):** senza generare una condizione di *STOP* intermedia, il master genera una nuova condizione di *START* in modalità lettura (*repeated START*, 1 bit), reinvia l'indirizzo dello slave con bit di lettura e relativo ACK (9 bit) e riceve sequenzialmente i byte di dato, ciascuno codificato su 9 bit (8 bit di dato più ACK/NACK).

Al termine della ricezione, viene generata la condizione di *STOP* (1 bit), che conclude la transazione. Nel sistema in esame, per ciascuna lettura vengono acquisiti i tre assi del sensore selezionato (accelerometro oppure giroscopio), ciascuno codificato su 16 bit. Ne consegue che il payload trasferito in una singola operazione di lettura è pari a:

$$N_{data} = 3 \cdot 2 = 6 \text{ byte.}$$

Il numero complessivo di bit trasferiti durante una singola operazione di *memory read* può essere stimato come:

$$N_{bit} = 1 + 9 + 9 + 1 + 9 + 9N_{data} + 1 = 30 + 9N_{data},$$

dove i termini rappresentano rispettivamente le condizioni di *START*, indirizzo con bit di scrittura, indirizzo del registro interno, *repeated START*, indirizzo con bit di lettura, dati e condizione di *STOP*.

Sostituendo  $N_{data} = 6$ , si ottiene:

$$N_{bit} = 30 + 9 \cdot 6 = 84 \text{ bit.}$$

Il tempo necessario a completare una singola lettura (accelerometro *oppure* giroscopio) risulta quindi:

$$C_{I^2C,MPU,ACCEL} = C_{I^2C,MPU,GYRO} = \frac{84}{100\,000} \text{ s} = 0.84 \text{ ms.}$$

Poiché, ad ogni attivazione del task **supervisionTask**, vengono effettuate due operazioni di lettura distinte (una per l'accelerometro e una per il giroscopio), il tempo complessivo associato all'acquisizione dei dati inerziali risulta:

$$C_{I^2C,MPU} = 2 \cdot 0.84 \text{ ms} \approx 1.68 \text{ ms.}$$

**WCET per la lettura dell' I<sup>2</sup>C** Il tempo di esecuzione complessivo per la lettura delle periferiche I<sup>2</sup>C si ottiene sommando i contributi temporali delle due acquisizioni eseguite ad ogni attivazione: lettura del ricevitore del controller e lettura dei dati inerziali dal MPU6050. Pertanto:

$$C_{readI2CSensors} = C_{I^2C,RX} + C_{I^2C,MPU} = 1.55 \text{ ms} + 1.68 \text{ ms} = 3.23 \text{ ms.}$$

A conferma di quest'analisi viene mostrata una stima condotta sperimentalmente in figura 8.1.

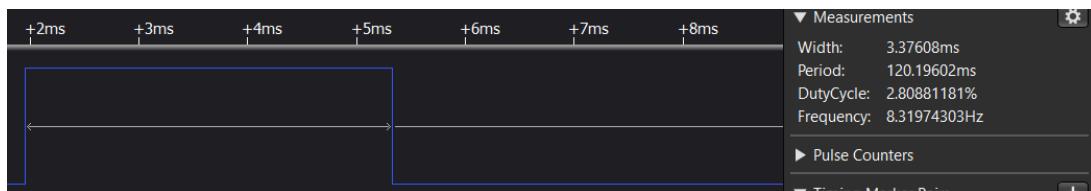


Figura 8.1: Misura del tempo di esecuzione per la lettura delle periferiche I<sup>2</sup>C sulla Board 2

Il valore sperimentale viene assunto come WCET per la lettura delle periferiche I<sup>2</sup>C in quanto include gli overhead non modellati nella stima analitica:

$$C_{readI2CSensors} = 3.38 \text{ ms},$$

**Parte di supervisione** Per la Board 2, che agisce come iniziatore del ciclo, il task di supervisione è modellato come un Task Periodico ( $\tau_{sup}$ ). I parametri temporali sono definiti come segue:

- Periodo ( $T$ ): 60 ms
- Deadline ( $D$ ): Coincidente con il periodo ( $D = T = 60$  ms).

La scelta di questo valore non è arbitraria, ma deriva da una relazione con il task di acquisizione dei sensori a ultrasuoni (Sonar). Impostando il periodo di supervisione come sottomultiplo del ciclo di scansione sonar (120 ms), si ottiene una schedulazione ottimizzata: il sistema garantisce un aggiornamento dei dati ambientali ogni due cicli di controllo. Questo bilanciamento assicura un'elevata reattività ai comandi di navigazione pur mantenendo la gestione dei sensori sicura e consistente. Il Worst Case Execution Time del task di supervisione sulla Board 2 è stato già discusso nel paragrafo 8.4.1.

**Attuazione (anello aperto) all'interno di *supervisionTask*** Coerentemente con l'organizzazione adottata sulla Board 2, la fase di attuazione in anello aperto non è implementata come task periodico separato, ma viene eseguita direttamente all'interno della *supervisionTask* come fase conclusiva del ciclo di supervisione. In tale fase, le decisioni prodotte dalla logica di supervisione vengono tradotte in comandi fisici verso gli attuatori, garantendo una relazione temporale deterministica tra: acquisizione degli ingressi, elaborazione decisionale e applicazione dei comandi.

Il comando dei motori avviene tramite il driver di potenza *Sabertooth 2x12*, configurato in *Analog Input Mode*. In tale modalità, i comandi di velocità e direzione sono forniti sotto forma di segnali analogici su due ingressi distinti: una tensione di riferimento pari a 2.5 V corrisponde alla condizione di arresto, mentre valori superiori o inferiori a tale soglia comandano rispettivamente il moto in avanti o all'indietro. Il driver provvede internamente alla conversione di tali tensioni in segnali PWM per il pilotaggio dei motori.

La *supervisionTask* gestisce inoltre il relè di potenza del sistema, garantendo l'abilitazione o la disabilitazione dell'attuazione in funzione della modalità operativa e delle condizioni di sicurezza.

Al fine di stimare il tempo di esecuzione della fase di attuazione, è stata effettuata una misura sperimentale mediante l'utilizzo di un GPIO ausiliario, commutato in ingresso e in uscita dalla sezione di codice dedicata all'output. La misura ottenuta è riportata in Figura 8.2:

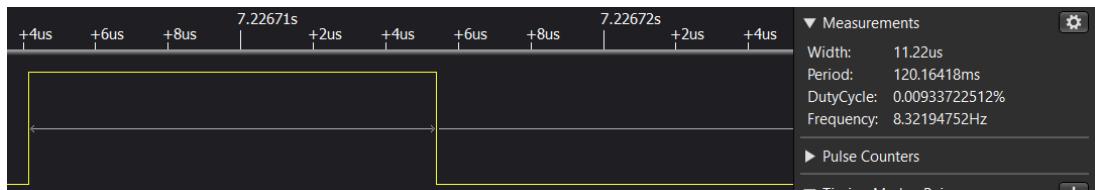


Figura 8.2: Misura del tempo di esecuzione della fase di attuazione (output) sulla Board 2

Il tempo di esecuzione misurato per la fase di attuazione risulta pari a:

$$C_{\text{output}} \approx 12 \mu\text{s},$$

valore che viene assunto come stima del Worst Case Execution Time del contributo di output all'interno della *supervisionTask* ai fini dell'analisi di schedulabilità.

**WCET complessivo e utilizzazione della *supervisionTask*** Il Worst Case Execution Time complessivo della *supervisionTask* si ottiene sommando i contributi delle diverse fasi che compongono il ciclo di supervisione sulla Board Master, ovvero:

- acquisizione degli ingressi tramite bus I<sup>2</sup>C;
- esecuzione della logica di supervisione e del protocollo di comunicazione inter-board;
- fase di attuazione in anello aperto.

Dalle analisi precedenti, tali contributi risultano pari a:

$$\begin{aligned}C_{I^2C} &= 3.38 \text{ ms}, \\C_{SUP} &= 35.4 \text{ ms}, \\C_{output} &= 12 \mu\text{s}.\end{aligned}$$

Il Worst Case Execution Time complessivo della *supervisionTask* può quindi essere espresso come:

$$\begin{aligned}C_{supervisionTask} &= C_{I^2C} + C_{SUP} + C_{output} \\&= 3.38 \text{ ms} + 35.4 \text{ ms} + 0.012 \text{ ms} \\&\approx 38.792 \text{ ms}.\end{aligned}$$

Il periodo di attivazione della *supervisionTask* è stato fissato pari a:

$$T_{supervisionTask} = 60 \text{ ms},$$

valore che coincide con la deadline del task ed è stato scelto come multiplo del periodo del task di lettura sonar, al fine di migliorare la prevedibilità della schedulazione.

A partire dal tempo di esecuzione stimato, l'utilizzazione della *supervisionTask* risulta:

$$U_{supervisionTask} = \frac{C_{supervisionTask}}{T_{supervisionTask}} = \frac{38.792 \text{ ms}}{60 \text{ ms}} \approx 0.6465.$$

### readSonarTask

Il **readSonarTask** è responsabile dell'interfacciamento con il modulo **HC-SR04**, il quale è un sensore a ultrasuoni che misura la distanza di un oggetto sfruttando il principio del *Time-of-Flight* (ToF).

Il ciclo di misura si svolge nel seguente modo:

- Il microcontrollore genera su **TRIG** un impulso alto di durata pari ad almeno  $10 \mu\text{s}$ .
- Il modulo emette un burst ultrasonico (tipicamente 8 cicli a 40 kHz) e porta il pin **ECHO** a livello alto.
- Se l'onda riflessa da un ostacolo viene ricevuta dal modulo, **ECHO** viene riportato basso.
- La durata dell'impulso alto su **ECHO**, indicata con  $t_{echo}$ , rappresenta il *Time-of-Flight* di andata e ritorno.

La latenza massima della misura non dipende dal microcontrollore, ma dalla fisica del fenomeno: maggiore è la distanza, maggiore è il tempo necessario affinché l'eco ritorni. Nel caso in cui l'oggetto si trovi alla distanza massima misurabile dal sensore, pari a  $d_{max} = 4 \text{ m}$ , il tempo massimo dell'impulso **ECHO** risulta:

$$t_{echo,max} \approx \frac{2d_{max}}{v} = \frac{2 \cdot 4}{340} \text{ s} \approx 23.5 \text{ ms}.$$

Quindi una singola misura può richiedere fino a circa 24 ms. Nel caso peggiore in assoluto, il sonar potrebbe non vedere nessun ostacolo: è importante chiarire che anche in questo caso il pin ECHO non rimane indefinitamente a livello alto. Il modulo HC-SR04, infatti, implementa internamente una logica di timeout: se l'eco non ritorna entro un certo intervallo temporale, ECHO viene comunque riportato a livello basso.

Nel rover sono presenti tre sensori HC-SR04 (sinistro, centrale, destro). In un sistema multi-sonar possono verificarsi due problemi principali:

- **Sovrapposizione temporale di misure sullo stesso sensore:** se si invia un nuovo TRIG prima che l'eco della misura precedente sia terminata, ECHO può riferirsi a echi “vecchi” o a condizioni miste.
- **Crosstalk tra sensori diversi:** il ricevitore di un sonar può ricevere l'eco generata dal burst emesso da un altro sonar vicino, producendo misure errate.

Per ridurre la probabilità di interferenze, è buona pratica distanziare le misure e, soprattutto in presenza di più sonar, evitare trigger simultanei.

Pertanto una strategia *naive* per la lettura consiste nell'attivare un sensore alla volta e attendere fino alla fine della misura, introducendo ritardi fissi dell'ordine di decine di millisecondi tra una misura e quella del sensore successivo. Tale approccio, sebbene funzionalmente corretto, risulta inconveniente nel nostro sistema poiché rende la fase di lettura di questi sensori estremamente dispendiosa in termini temporali.

L'obiettivo è stato quindi evitare l'attesa attiva durante la finestra in cui ECHO rimane alto, sfruttando meccanismi hardware che consentissero di acquisire l'informazione temporale senza occupare inutilmente la CPU. In particolare, la misura del Time-of-Flight è stata demandata a un timer hardware configurato in modalità *Input Capture*, in grado di generare interrupt sui fronti di salita e discesa del segnale ECHO. In questo modo, una volta avviata la misura tramite il segnale TRIG, il task di lettura sensori termina, lasciando la CPU disponibile per l'esecuzione di altri task.

Dal punto di vista dello scheduling, la soluzione adottata comporta che la fase computazionalmente più onerosa della misura non venga eseguita all'interno del task, bensì sia distribuita su due interrupt estremamente brevi, generati rispettivamente sul fronte di salita e sul fronte di discesa del segnale ECHO. Al fine di stimare il Worst Case Execution Time associato alla gestione degli eventi sonar, è stata effettuata una misura sperimentale della durata degli interrupt tramite analizzatore logico. In particolare, la durata di esecuzione delle ISR associate ai fronti di salita e di discesa del segnale ECHO è stata misurata utilizzando un segnale GPIO ausiliario, commutato all'ingresso e all'uscita di ciascuna routine di interrupt (Figura 8.3).

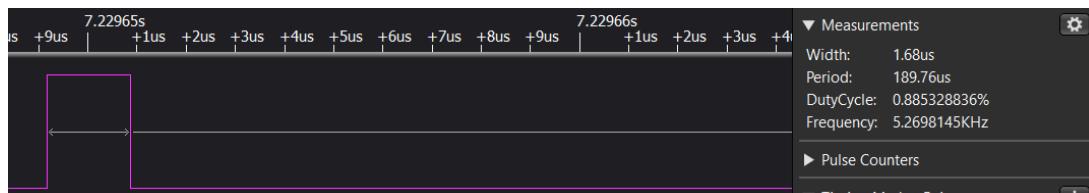


Figura 8.3: Misura del WCET degli interrupt associati al segnale ECHO del sensore HC-SR04

Dal punto di vista del modello di schedulazione, gli eventi generati dai sensori a ultrasuoni sono intrinsecamente *asincroni*, in quanto dipendono dall'istante di ritorno dell'eco, che a sua volta è funzione della distanza dell'ostacolo e non è determinabile a priori. In un modello teorico rigoroso, tali eventi devono quindi essere trattati come

*task sporadici*, caratterizzati da un tempo minimo di inter-arrivo e non da un periodo fisso. Tuttavia, ai fini dell'analisi di schedulabilità del sistema, è possibile ricondurre il comportamento dei sonar a un modello equivalente di task periodico facendo riferimento a una trasformazione conservativa comunemente adottata nei sistemi real-time. Il principio alla base di tale trasformazione consiste nell'identificare il tempo minimo di inter-arrivo  $T_{min}$  tra due attivazioni successive dell'evento sporadico e modellare quest'ultimo come un task periodico avente periodo  $T = T_{min}$ .

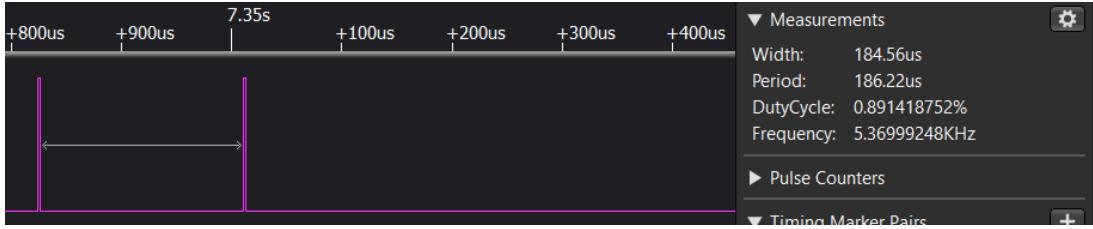


Figura 8.4: Intervallo minimo di inter-arrivo tra due interrupt consecutivi del segnale ECHO del sensore HC-SR04

In Figura 8.4 è riportata una cattura rappresentativa dell'intervallo temporale osservabile tra i due interrupt associati al fronte di salita e discesa di ECHO. La misura è stata condotta posizionando deliberatamente l'oggetto bersaglio alla distanza minima misurabile dal sensore, pari a circa 2 cm.

Dalle misure sperimentali riportate in Figura 8.3, la durata massima di esecuzione di ciascuna ISR associata al fronte di salita o di discesa del segnale ECHO risulta pari a:

$$C_{ISR} \approx 1.7 \mu s.$$

Dalla cattura mostrata in Figura 8.4, l'intervallo minimo di inter-arrivo tra due interrupt consecutivi è stimato in:

$$T_{min} \approx 184 \mu s.$$

Sostituendo tali valori nell'espressione dell'utilizzazione equivalente dell'evento sporadico:

$$U_{ISRSonar} = \frac{2 \cdot C_{ISR}}{T_{min}},$$

si ottiene:

$$U_{ISRSonar} = \frac{2 \cdot 1.7 \mu s}{184 \mu s} \approx 0.018.$$

L'utilizzazione equivalente risulta significativamente inferiore rispetto ai contributi dei task periodici principali. Per tale motivo, e al solo fine di semplificare l'analisi di schedulabilità globale, si è deciso di non includere esplicitamente  $U_{ISRSonar}$  nel calcolo finale dell'utilizzazione del sistema.

Per quanto riguarda il periodo di esecuzione del task di lettura vero e proprio dei sonar, è stato scelto un valore pari a  $T = 40$  ms. Dal punto di vista del modello di schedulazione, tale scelta garantisce che ogni attivazione del task avvenga quando la precedente misura è certamente conclusa, anche nel caso peggiore di distanza massima o di timeout interno del sensore. Il task di lettura può quindi essere modellato come un task periodico con periodo fissato e noto. Il tempo di esecuzione  $T_{readSonarTask}$  del task di lettura è stato quindi determinato sperimentalmente. Anche in questo caso la misura è stata effettuata

acquisendo un segnale di GPIO ausiliario, riportata in Figura 8.5.

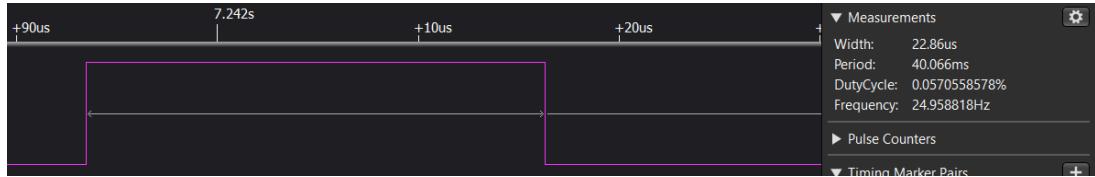


Figura 8.5: Tempo di esecuzione del task di lettura sonar

La durata di esecuzione del task di lettura sonar risulta pari a:

$$C_{readSonarTask} \approx 23 \mu s.$$

Il periodo del task è stato fissato a:

$$T_{readSonarTask} = 40 \text{ ms.}$$

L'utilizzazione del task di lettura sonar può quindi essere calcolata come:

$$U_{readSonarTask} = \frac{C_{readSonarTask}}{T_{readSonarTask}} = \frac{23 \mu s}{40 \text{ ms}} \approx 5.7 \times 10^{-4}.$$

### telemetryLoggerTask

Il `telemetryLoggerTask` è responsabile della trasmissione periodica della struttura `Telemetry_t` verso il nodo ESP32-S3 tramite bus I<sup>2</sup>C. La comunicazione avviene in modalità *Standard Mode* con frequenza:

$$f_{I^2C} = 100 \text{ kHz.}$$

**Dimensione del payload** La struttura `Telemetry_t`, dichiarata come `__attribute__((packed))`, ha dimensione complessiva:

$$N_{data} = 24 \text{ byte.}$$

Nel protocollo I<sup>2</sup>C, ogni byte richiede 9 bit (8 bit dati + ACK), pertanto il tempo necessario alla trasmissione del solo payload risulta:

$$T_{data} = \frac{24 \cdot 9}{100\,000} = \frac{216}{100\,000} = 2.16 \text{ ms.}$$

**Overhead di protocollo** Alla trasmissione dei dati si aggiungono:

- indirizzo slave (7 bit + R/W + ACK): 9 bit;
- condizioni di START e STOP: 2 bit.

Il contributo temporale associato è quindi:

$$T_{addr} = \frac{9}{100\,000} = 90 \mu s,$$

$$T_{START+STOP} = \frac{2}{100\,000} = 20 \mu s.$$

**WCET analitico della trasmissione** Il Worst Case Execution Time della trasmissione su I<sup>2</sup>C risulta:

$$\begin{aligned} C_{I^2C,TX} &= T_{addr} + T_{data} + T_{START+STOP} \\ &= 90 \mu s + 2.16 \text{ ms} + 20 \mu s \\ &= 2.27 \text{ ms}. \end{aligned}$$

La preparazione della struttura `Telemetry_t` consiste in:

- copia atomica delle variabili condivise;
- assegnazioni dirette ai campi della struttura locale della board.

Il contributo temporale associato a queste operazioni è trascurabile rispetto alla trasmissione su bus e viene modellato come un termine  $\varepsilon$  tale che:

$$\varepsilon \ll 1 \text{ ms}.$$

Pertanto:

$$C_{\text{telemetryLoggerTask}} = C_{I^2C,TX} + \varepsilon \approx 2.30 \text{ ms}.$$

Il task è modellato come task periodico con:

$$T_{\text{telemetryLoggerTask}} = 120 \text{ ms},$$

coincidente a due periodi della `supervisionTask`, in modo da garantire un aggiornamento coerente della telemetria ad due cicli decisionale.

L'utilizzazione del task risulta pertanto:

$$U_{\text{telemetryLoggerTask}} = \frac{C_{\text{telemetryLoggerTask}}}{T_{\text{telemetryLoggerTask}}} = \frac{2.30 \text{ ms}}{120 \text{ ms}} \approx 0.019.$$

### 8.4.3 Board 1

#### supervisionTask (Board 1)

Per la Board 1, l'analisi richiede un approccio differente rispetto alla Board 2. Essendo il sistema privo di una linea di clock condivisa, le due board evolvono basandosi su oscillatori locali indipendenti. Modellare anche la Board 1 come task periodico introducerebbe inevitabilmente il problema del Clock Drift: le derive temporali dei due clock porterebbero, nel lungo periodo, a una perdita di sincronizzazione.

**Funzionamento Nominale** In condizioni normali, l'esecuzione è strettamente asservita al ciclo della Board 2:

- **Trigger di Attivazione:** Il task permane in uno stato di attesa finché non viene rilevato il fronte di salita (rising edge) del segnale fisico SESSION. Questo evento hardware imposta un event flag che sblocca immediatamente l'esecuzione del task, garantendo la sincronizzazione di fase.
- **Minimum Inter-arrival Time (MIT):** Poiché l'evento di attivazione è generato dal Master che opera con una periodicità fissa, il tempo minimo di interarrivo tra due attivazioni consecutive corrisponde al periodo del Master stesso ( $MIT = 60\text{ ms}$ ).

L'adozione del modello sporadico su Board 1 implica che l'esecuzione sia asservita ai tempi del Master. Nello scenario peggiore, i dati locali acquisiti dalla Board 1 potrebbero non essere perfettamente sincroni con l'inizio del ciclo di comunicazione. Tuttavia, tale latenza è ritenuta accettabile per la natura delle variabili gestite dal Slave:

1. Variabili a dinamica lenta: Parametri come la temperatura interna e il voltaggio della batteria non subiscono variazioni significative nell'ordine dei millisecondi.
2. Monitoraggio Motori: Le velocità dei motori, pur essendo variabili dinamiche, vengono utilizzate nel contesto della supervisione principalmente per controlli di sicurezza binari (es. verificare se il rover è fermo prima di cambiare stato).

Pertanto, l'architettura garantisce la sicurezza operativa anche in presenza di lievi disallineamenti temporali nell'acquisizione dello Slave.

**Gestione del Guasto** Il modello prevede una logica di robustezza per gestire la perdita del segnale di sincronizzazione (es. guasto della linea SESSION o crash del Master). Il task attende l'evento esterno impostando un timeout coincidente con la periodicità attesa (60 ms). Qualora l'evento non si verifichi entro tale limite, il sistema evolve come segue:

- Risveglio per Timeout: Il task viene sbloccato dal sistema operativo non per l'occorrenza dell'evento esterno, ma per la scadenza del timer interno.
- Transizione in Degradato: Il firmware esegue lo step della macchina a stati che, rilevando l'assenza di una sessione valida avviata dal Master, forza la transizione nello stato DEGRADATO.
- Ciclo Autonomo: Da questo momento, in assenza dello stimolo esterno, la Board 1 opera in autonomia temporale. Essa si risveglia ciclicamente alla scadenza del timeout (ogni 60 ms) per eseguire l'algoritmo decisionale locale.

- Risincronizzazion: Questa modalità operativa persiste finché non viene nuovamente rilevato un rising edge sulla linea SESSION. L'arrivo dell'evento hardware ha priorità assoluta sul timeout: esso risveglia immediatamente il task e riallinea la Board 1 al ciclo del Master, permettendo al sistema di abbandonare la modalità degradata e riprendere la cooperazione nominale.

Il Worst Case Execution Time del task di supervisione sulla Board 1 è stato già discusso nel paragrafo 8.4.1.

### **readSensorTask**

Il **readSensorTask** è un task periodico del sistema real-time responsabile dell'acquisizione dei dati provenienti dai principali sensori collegati alla *Board 1*. Il suo obiettivo è fornire al sistema una visione aggiornata e coerente dello stato fisico del dispositivo, separando chiaramente la fase di acquisizione dei dati dalla loro elaborazione da parte di altri task.

In particolare, il task si occupa di:

- leggere i segnali dagli encoder, uno per ogni motore, acquisendo i canali CH1 e CH2 necessari alla determinazione della velocità;
- acquisire la tensione della batteria tramite il relativo canale analogico, per monitorare lo stato di alimentazione del sistema;
- leggere il sensore interno di temperatura del microcontrollore, utilizzato per il monitoraggio termico e la diagnostica.

**Acquisizione degli encoder** La lettura degli encoder avviene tramite degli encoder incrementali a due canali. Ad ogni attivazione del task, viene campionato il valore corrente del contatore associato a ciascun encoder e confrontato con il valore memorizzato al ciclo precedente. La differenza tra le due letture rappresenta il numero di impulsi generati nell'intervallo di campionamento. Il segno della differenza consente di determinare il verso di rotazione del motore. Eventuali condizioni di overflow del contatore sono gestite esplicitamente per garantire la continuità della misura anche in presenza di rotazioni prolungate. A partire dal numero di impulsi rilevati, il task calcola una stima della velocità angolare del motore, normalizzando il dato rispetto al periodo di campionamento, alla risoluzione dell'encoder e al rapporto di riduzione meccanico. Il valore finale viene discretizzato e convertito in formato intero per l'utilizzo nei task di controllo.

**Misura della tensione di batteria** La tensione della batteria viene acquisita mediante il convertitore analogico-digitale (ADC) del microcontrollore. Il segnale analogico è prelevato a valle di un partitore resistivo, che consente di riportare la tensione entro il range ammesso dall'ADC. Il valore digitale ottenuto dalla conversione viene convertito in una tensione equivalente e successivamente compensato per ricostruire la tensione reale della batteria. A partire da tale valore viene inoltre calcolata una stima approssimata dello stato di carica mediante una mappatura lineare tra una tensione minima e una massima di riferimento, con saturazione agli estremi per evitare risultati non fisicamente significativi.

**Misura della temperatura interna** La temperatura del microcontrollore viene misurata tramite il sensore di temperatura interno, collegato internamente all'ADC. Anche in questo caso viene eseguita una singola conversione per ogni attivazione del task. Il valore digitale viene convertito in temperatura assoluta applicando la relazione lineare basata sui

parametri di calibrazione forniti dal costruttore. La misura è utilizzata esclusivamente per monitoraggio e diagnostica e non interviene direttamente nelle decisioni di controllo a basso livello.

I dati prodotti dal task vengono pubblicati su una struttura globale condivisa, accessibile anche da altri task con priorità e periodicità differenti. Poiché l'architettura software è basata su multitasking preemptive, l'aggiornamento di tali dati può avvenire in concorrenza con la loro lettura da parte di altri task. Per prevenire condizioni di race e garantire che i valori osservati siano sempre coerenti e atomicamente aggiornati, l'accesso in scrittura alla struttura condivisa è protetto tramite una sezione critica come discusso in dettaglio nella Sezione 8.3.

Il `readSensorTask` viene eseguito con una periodicità fissa pari a

$$T_{\text{readSensorTask}} = 120 \text{ ms},$$

scelta in funzione della dinamica dei segnali da acquisire e dei requisiti temporali del sistema. In Figura 8.6 è riportato il tempo di esecuzione del task, misurato sperimentalmente e pari a:

$$C_{\text{readSensorTask}} \approx 1.15 \text{ ms},$$

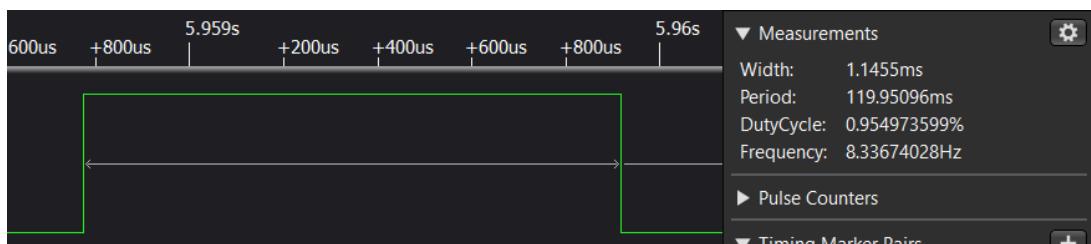


Figura 8.6: Tempo di esecuzione del task `readSensorTask`

L'utilizzazione del task risulta:

$$U_{\text{readSensorTask}} = \frac{C_{\text{readSensorTask}}}{T_{\text{readSensorTask}}} = \frac{1.15 \text{ ms}}{120 \text{ ms}} \approx 9.6 \times 10^{-3}.$$

### `pidTask`

Il `pidTask` è un task periodico responsabile dell'esecuzione del ciclo di controllo in retroazione dei quattro motori della *Board 1*. Il suo compito principale è trasformare i riferimenti di velocità prodotti dai livelli superiori in comandi attuativi per i motori, compensando l'errore tra riferimento e velocità misurata tramite un regolatore PID discreto con saturazione. Il task viene eseguito con periodicità fissa pari a  $T_{\text{pidTask}} = 5 \text{ ms}$ , che rappresenta il tempo di campionamento del controllo. Tale scelta impone che l'intero ciclo di esecuzione del task venga completato entro il periodo, al fine di garantire la corretta evoluzione del sistema di controllo in tempo discreto.

Ad ogni attivazione, il `pidTask`:

- acquisisce in modo atomico i riferimenti di velocità e la modalità di frenata tramite accesso protetto;
- legge la velocità effettiva dei quattro motori mediante encoder;
- applica una rampa ai riferimenti per limitare variazioni brusche e gestire in modo differenziato la frenata normale e quella di emergenza;

- calcola, per ciascun motore, l'azione di controllo mediante una legge PID discreta con saturazione;
- converte l'azione di controllo in comandi PWM e aggiorna le uscite hardware di pilotaggio.

La rampa applicata ai riferimenti consente di limitare le sollecitazioni meccaniche ed elettriche in condizioni nominali e di ridurre la latenza della manovra in caso di frenata di emergenza, consentendo l'applicazione immediata del riferimento nullo quando necessario. In Figura 8.7 è riportato il tempo di esecuzione del `pidTask`, misurato sperimentalmente.

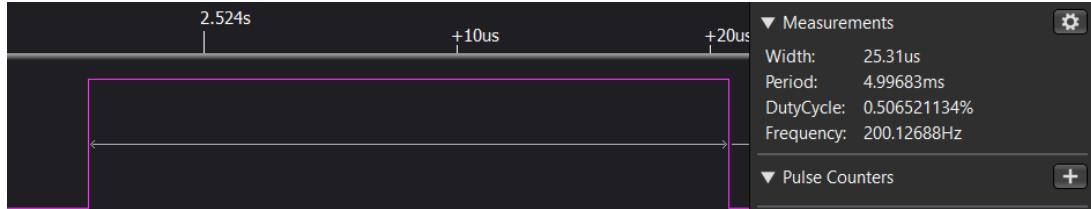


Figura 8.7: Tempo di esecuzione del task `pidTask`

A partire dal tempo di esecuzione misurato:

$$C_{\text{pidTask}} = 26 \mu\text{s},$$

l'utilizzazione del task risulta:

$$U_{\text{pidTask}} = \frac{C_{\text{pidTask}}}{T_{\text{pidTask}}} = \frac{26 \mu\text{s}}{5 \text{ ms}} \approx 5.2 \times 10^{-3}.$$

Ancora una volta, eventuali condizioni di contesa sulle risorse condivise sono trattate con le procedure descritte nella Sezione 8.3.

### **lightsTask**

Il `lightsTask` è un task periodico incaricato di attuare le decisioni prese dal livello di supervisione relativamente alla segnalazione luminosa del veicolo. Il task gestisce due tipologie di attuatori distinti:

- due LED anteriori discreti, comandati tramite operazioni di set/reset su GPIO;
- due strisce LED RGB indirizzabili, pilotate mediante timer PWM e trasferimenti DMA.

Ad ogni attivazione, il `lightsTask` esegue le seguenti operazioni:

- aggiorna lo stato dei LED discreti mediante operazioni di set/reset su GPIO;
- aggiorna lo stato logico delle animazioni delle strisce LED e, quando necessario, avvia la trasmissione del frame tramite timer PWM e DMA.

L'accesso alle strutture dati condivise avviene esclusivamente all'interno di sezioni critiche di durata estremamente limitata. Data la brevità delle operazioni protette, tale scelta non introduce attese bloccanti né impatti significativi sulla schedulazione del sistema, risultando coerente con quanto discusso nella Sezione 8.3.

**Pilotaggio delle strisce LED** Le strisce LED posteriori sono di tipo RGB indirizzabile e richiedono un bitstream con vincoli temporali stringenti. Per evitare attese attive e rendere l’aggiornamento indipendente dal carico della CPU, la generazione del segnale è demandata alla combinazione **timer PWM + DMA**.

Dal punto di vista software, il contenuto cromatico dei pixel viene mantenuto in un buffer lineare (`rgb_arr`), aggiornato dal `lightsTask` in base all’animazione selezionata. Quando è necessario inviare un nuovo frame, il task invoca `led_render()`, che:

- inizializza la trasmissione predisponendo il primo chunk del buffer PWM (`wr_buf`);
- avvia il trasferimento DMA verso il registro del timer tramite `HAL_TIM_PWM_Start_DMA()`.

La trasmissione dell’intero frame non viene preparata in un’unica soluzione: durante l’invio, le routine di callback del DMA riempiono progressivamente le metà del buffer `wr_buf` (meccanismo half-transfer e transfer-complete), convertendo i byte RGB in sequenze PWM *high/low* coerenti con il protocollo della striscia. Terminati i pixel, le callback generano inoltre il tratto di *reset* mediante inserimento di zeri, completando correttamente il frame prima di arrestare la periferica.

Se una nuova richiesta di aggiornamento arriva durante una trasmissione già in corso (DMA *busy*), la funzione `led_render()` abortisce il trasferimento corrente, azzera il buffer PWM e re-inizializza la sequenza, evitando che frame parziali producano effetti visivi incoerenti. In Figura 8.8 è riportata una misura rappresentativa del tempo di esecuzione del task.

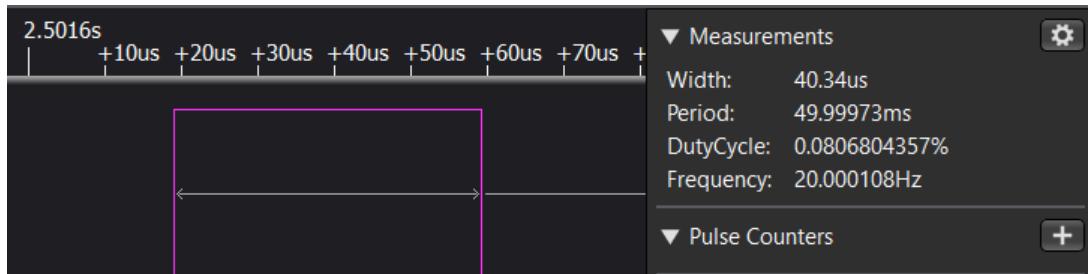


Figura 8.8: Tempo di esecuzione del task `lightsTask`

A partire dalla misura sperimentale, il tempo di esecuzione del task viene assunto come:

$$C_{\text{lightsTask}} \approx 41 \mu\text{s}.$$

Il task è eseguito con una **periodicità fissa pari a**:

$$T_{\text{lightsTask}} = 50 \text{ ms},$$

valore scelto per garantire animazioni luminose visivamente fluide e coerenti con la dinamica del veicolo, senza introdurre un carico eccessivo sul processore.

L’utilizzazione del task risulta:

$$U_{\text{lightsTask}} = \frac{C_{\text{lightsTask}}}{T_{\text{lightsTask}}} = \frac{41 \mu\text{s}}{50 \text{ ms}} = 8.2 \times 10^{-4}.$$

**Nota sulle callback DMA** Le callback DMA associate alla trasmissione delle strisce LED (half-transfer e transfer-complete) sono eseguite in contesto di interrupt e non fanno parte del thread periodico del `lightsTask`.

Analogamente a quanto fatto per gli interrupt associati ai sensori a ultrasuoni nel paragrafo

8.4.2, ciascuna callback DMA è stata modellata come un task sporadico e il suo tempo di esecuzione è stato caratterizzato mediante misura sperimentale. In questo caso, tuttavia, la frequenza massima di arrivo degli interrupt è vincolata dall'attivazione del `lightsTask`: per ogni attivazione del task può essere generata al più una sequenza di callback DMA associata alla trasmissione di un frame LED.

Di conseguenza, ai fini dell'analisi di schedulabilità, le callback DMA possono essere ricondotte a un modello equivalente di task periodico avente periodo pari a quello del `lightsTask`:

$$T_{\text{ISR,DMA}} = T_{\text{lightsTask}} = 50 \text{ ms}.$$

Dalle misure sperimentali effettuate, il tempo di esecuzione di ciascuna ISR DMA risulta pari a:

$$C_{\text{ISR,DMA}} \approx 10 \mu\text{s}.$$

Poiché tali interrupt non vengono eseguiti nel contesto del thread periodico, il loro contributo non è incluso nel WCET del `lightsTask`, ma considerato separatamente come carico IRQ del sistema. L'utilizzazione del ISR risulta pertanto:

$$U_{\text{ISR,DMA}} = \frac{C_{\text{ISR,DMA}}}{T_{\text{ISR,DMA}}} = \frac{10 \mu\text{s}}{50 \text{ ms}} = 2.0 \times 10^{-4}.$$

Tale contributo risulta trascurabile rispetto all'utilizzazione complessiva dei task periodici principali; pertanto, al solo fine di semplificare l'analisi di schedulabilità globale, si è scelto di non includere esplicitamente  $U_{\text{ISR,DMA}}$  nel calcolo finale dello scheduling.

## 8.5 Scheduling e dimostrazione di schedulabilità

Data la natura fortemente periodica dei task individuati, si è deciso di impiegare una politica di scheduling **Rate Monotonic** (RM) per entrambe le board. Tale scelta è giustificata dall'ottimalità dell'algoritmo per sistemi a priorità fissa e dalla sua natura statica, che riduce l'overhead computazionale dell'RTOS rispetto ad algoritmi dinamici. Inoltre, l'assegnazione delle priorità basata sulla frequenza risponde ai requisiti di controllo del rover, garantendo un comportamento temporale prevedibile.

Sulla base del task set individuato e dei tempi di esecuzione stimati nei paragrafi precedenti, è ora possibile procedere alla dimostrazione della schedulabilità del sistema real-time implementato sul rover. L'analisi viene condotta separatamente per ciascuna board, in quanto le due unità eseguono insiemi di task distinti su microcontrollori dedicati e non condividono risorse di calcolo. Ciascuna board viene modellata come un insieme di task periodici indipendenti, dove ogni task  $\tau_i$  è caratterizzato da un **Worst Case Execution Time**  $C_i$ , da un **periodo** di attivazione  $T_i$  e da una **deadline** relativa  $D_i$ . Nel sistema in esame, i task sono implementati in maniera tale che la deadline relativa coincida con il periodo di attivazione:

$$D_i = T_i.$$

Questa assunzione è coerente con la natura dei task di controllo e supervisione e rappresenta una scelta conservativa ai fini dell'analisi. Con politica RM la priorità dei task è assegnata in funzione inversa del periodo:

$$T_i < T_j \Rightarrow \text{prio}(\tau_i) > \text{prio}(\tau_j).$$

Una prima verifica della schedulabilità del sistema viene effettuata tramite il test di utilizzazione di Liu e Layland, che fornisce una **condizione sufficiente** per la schedulabilità sotto RM. Dato un insieme di  $n$  task periodici, la condizione è espressa come:

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq n \left( 2^{1/n} - 1 \right),$$

dove il termine a destra rappresenta il limite superiore teorico dell'utilizzazione ammissibile per un sistema schedulato con Rate Monotonic.

### 8.5.1 Scheduling della Board 1

Come descritto nelle sezioni precedenti, la **Board 1** esegue un insieme di **quattro task periodici** dedicati al controllo in retroazione, al monitoraggio sensoristico, alla gestione dell'illuminazione e alla supervisione. In Tabella 8.1 sono riportati, per ciascun task  $\tau_i$ , il Worst Case Execution Time  $C_i$ , il periodo di attivazione  $T_i$  e il corrispondente fattore di utilizzazione  $U_i = C_i/T_i$ .

Tabella 8.1: Board 1: recap dei task periodici

Task	$C$	$T$	$U = C/T$
readSensorTask	1.15 ms	60 ms	$1,9 \times 10^{-2}$ .
pidTask	26 $\mu$ s	5 ms	$5.2 \times 10^{-3}$
lightTask	41 $\mu$ s	50 ms	$8.2 \times 10^{-4}$
supervisionTask	38.7 ms	60 ms	0.645
<b>Totale</b>			0.67

L'utilizzazione complessiva della CPU sulla Board 1 risulta quindi:

$$U_{B1} = \sum_i \frac{C_i}{T_i} = 0.67.$$

Poiché la Board 1 è schedulata con politica *Rate Monotonic* e il task set è composto da  $n = 4$  task periodici, si applica il bound di Liu e Layland, che fornisce una condizione sufficiente di schedulabilità:

$$U_{B1} \leq U_{RM}(4) = 4 \left( 2^{1/4} - 1 \right) \approx 0.756.$$

Dal confronto tra i valori ottenuti,  $U_{B1} = 0.67 < 0.756$ , si conclude che il task set della Board 1 è **schedulabile** sotto RM (condizione sufficiente), assumendo  $D_i = T_i$ .

Sulla base di tali considerazioni, in Figura 8.9 è riportato un diagramma di Gantt rappresentativo dello scheduling dei task sulla Board 1, che illustra l'ordine di esecuzione imposto dalle priorità Rate Monotonic e l'assenza di violazioni delle deadline all'interno dell'intervallo temporale considerato.

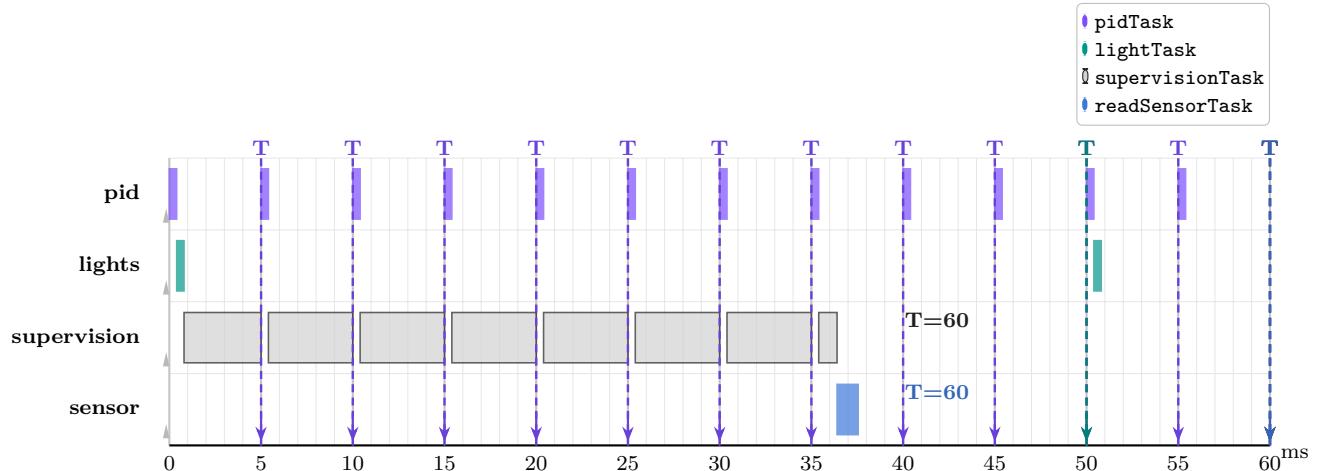


Figura 8.9: Scheduling RM sulla Board 1.

**Nota**

Per la **Board 1 (Slave)** l'intervallo temporale mostrato nel diagramma non coincide con un vero **iperperiodo** del sistema: la schedulazione prosegue oltre la finestra rappresentata e non si limita al termine dell'intervallo considerato.

Per **semplicità espositiva** ci si limita tuttavia a questa rappresentazione temporale; sebbene comunque, la **fattibilità della schedulazione** è comunque garantita alla luce del **test di Liu e Layland** effettuato e non emergono criticità di deadline nel funzionamento previsto.

### 8.5.2 Scheduling della Board 2

Come descritto nelle sezioni precedenti, la **Board 2** esegue un insieme di **due task periodici** dedicati alla lettura dei sonar e alla supervisione (comprensiva del protocollo di comunicazione inter-board). In Tabella 8.2 sono riportati, per ciascun task  $\tau_i$ , il Worst Case Execution Time  $C_i$ , il periodo di attivazione  $T_i$  e il corrispondente fattore di utilizzazione  $U_i = C_i/T_i$ .

Tabella 8.2: Board 2: recap dei task periodici

Task	$C$	$T$	$U = C/T$
readSonarTask	23 $\mu s$	40 ms	$5.7 \times 10^{-4}$
supervisionTask	39.7 ms	60 ms	0.6617
telemetryTxTask	2.30 ms	120 ms	0.019
<b>Totale</b>			0.682

L'utilizzazione complessiva della CPU sulla Board 2 risulta quindi:

$$U_{B2} = \sum_i \frac{C_i}{T_i} = 0.682.$$

Poiché la Board 2 è schedulata con politica *Rate Monotonic* e il task set è composto da  $n = 3$  task periodici, si applica il bound di Liu e Layland, che fornisce una condizione sufficiente di schedutabilità:

$$U_{RM}(3) = 3 \left( 2^{1/3} - 1 \right) \approx 0.779.$$

Dal confronto tra i valori ottenuti,  $U_{B2} = 0.682 < 0.779$ , si conclude che il task set della Board 2 è **schedulabile** sotto RM (condizione sufficiente), assumendo  $D_i = T_i$ .

Sulla base di tali considerazioni, in Figura 8.10 è riportato un diagramma di Gantt rappresentativo dello scheduling dei task sulla Board 2, che illustra l'ordine di esecuzione imposto dalle priorità Rate Monotonic e l'assenza di violazioni delle deadline all'interno dell'intervallo temporale considerato.

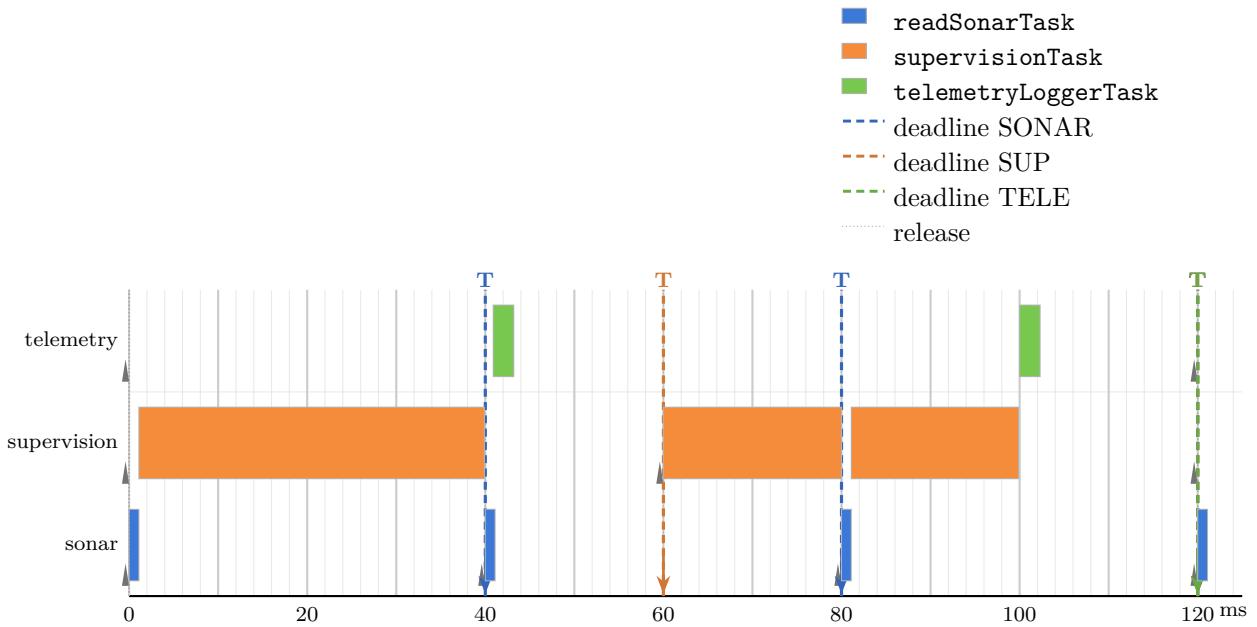


Figura 8.10: Diagramma di Gantt rappresentativo dello scheduling sulla Board 2 (RM)

### Vincolo temporale di sistema

Dall'analisi delle schedulazioni presentate si osserva che il sistema è in grado di operare nel rispetto della **hard deadline di 220 ms**. In particolare, la distribuzione temporale dei task e le finestre di esecuzione risultanti mostrano che tutte le attività critiche completano la propria esecuzione entro il vincolo temporale imposto. Tale deadline è stata definita a partire dalle considerazioni sullo spazio e sul tempo di frenata del rover, discusse nella sezione 8.1, e rappresenta un requisito fondamentale per garantire un comportamento sicuro del sistema in condizioni operative ordinarie. Le schedulazioni riportate forniscono quindi evidenza della coerenza tra le scelte di progetto a livello temporale e i requisiti fisici del sistema.

## 8.6 Stato degradato

Lo **stato degradato** rappresenta la modalità operativa in cui il rover deve mantenere un comportamento *deterministico e sicuro* anche in presenza di guasti che impediscono la cooperazione nominale tra le due board.

Per garantire tale continuità operativa, le logiche di gestione errore sono state **integrate direttamente all'interno dell'automa Stateflow**. In questo modo:

- non è necessario modificare il **task set** a runtime;
- la schedulazione RTOS resta **stazionaria** e quindi prevedibile;
- al verificarsi di un guasto, è la **macchina a stati** a deviare il flusso di esecuzione verso configurazioni e transizioni dedicate alla gestione della condizione anomala.

In funzionamento nominale la **supervisionTask** della **Board 1** è *event-driven*: viene attivata dal fronte di salita della linea SESSION, che ne impone implicitamente il ritmo (60 ms).

Se il fronte atteso non viene rilevato entro tale intervallo, l'automa transita in DEGRADED

e la supervisione diventa *time-driven*: il timeout di attesa si trasforma nel riferimento temporale e il task viene eseguito periodicamente ogni 60 ms in completa autonomia.

Il ritorno del segnale SESSION ha priorità sul timeout e riallinea immediatamente la Board 1 al ciclo del Master.

In degradato, quindi, cambia la **sorgente di attivazione** della supervisione (evento → timer), che ne scandisce l'esecuzione **ogni** 60 ms.

La **Board 2** esegue sempre la supervisionTask come **task periodico** con periodo  $T = 60$  ms. In condizioni nominali il ciclo include acquisizione I<sup>2</sup>C, cooperazione inter-board e attuazione.

In caso di guasto di comunicazione, l'automa transita in DEGRADED: la supervisione resta periodica, opera in modalità *single board* e invia **ping** a ogni ciclo per tentare il ripristino della connessione.

Dal punto di vista real-time non cambia nulla: la struttura temporale del task è invariata e il WCET dimensionato sul caso peggiore nominale rimane valido anche in degradato.

## 8.7 Dettagli implementativi in FreeRTOS

In questa sezione vengono descritti i principali dettagli implementativi relativi all'utilizzo di *FreeRTOS* all'interno del sistema. In particolare, sono illustrate e motivate le configurazioni adottate per la creazione e gestione dei task, l'organizzazione della memoria, l'assegnazione delle priorità e il controllo delle deadline, con l'obiettivo di garantire un comportamento deterministico e conforme ai requisiti temporali definiti in fase di progetto.

### 8.7.1 Allocazione statica dei task

Per quanto riguarda la gestione della memoria, tutti i task del sistema sono stati creati utilizzando l'allocazione **statica** prevista da FreeRTOS. In particolare, per ciascun task sono stati definiti esplicitamente lo stack e il *Task Control Block* (TCB), evitando completamente l'uso dell'allocazione dinamica a runtime.

Questa scelta è motivata da diverse considerazioni progettuali. In primo luogo, l'allocazione statica consente di eliminare il rischio di frammentazione dell'heap, fenomeno particolarmente critico nei sistemi embedded a lunga esecuzione e con risorse di memoria limitate. In secondo luogo, l'assenza di chiamate dinamiche a funzioni di allocazione durante il funzionamento del sistema migliora la prevedibilità temporale del software, riducendo le possibili sorgenti di jitter e di comportamento non deterministico.

Infine, l'allocazione statica permette di conoscere a priori l'occupazione di memoria dell'intero sistema, facilitando le attività di analisi, validazione e verifica dei requisiti di affidabilità, in linea con le buone pratiche di progettazione di sistemi real-time.

### 8.7.2 Assegnazione delle priorità e schedulazione

I task previsti dal sistema sono stati progettati e configurati assegnando a ciascuno una priorità specifica, coerente con i rispettivi vincoli temporali. In particolare, le priorità sono state scelte in modo da implementare una politica di schedulazione *Rate Monotonic*, come già descritto nelle sezioni precedenti.

Secondo questo criterio, ai task con periodo di attivazione più breve è stata assegnata una priorità più elevata, mentre i task con periodo maggiore sono stati collocati a priorità inferiori. In questo modo, lo scheduler di FreeRTOS — basato su priorità fisse e preemptive

— garantisce che i task più critici dal punto di vista temporale possano preemptare quelli meno critici, rispettando i vincoli di deadline assunti in fase di progetto.

L'adozione di questa strategia consente di ottenere un comportamento deterministico del sistema e di rendere applicabili le analisi teoriche di schedulabilità tipiche dei sistemi real-time a priorità fisse, assicurando coerenza tra il modello temporale descritto e l'implementazione effettiva su piattaforma STM32.

Board 1		Board 2	
Task	Priorità	Task	Priorità
supervisionTask	osPriorityNormal	supervisionTask	osPriorityNormal
readSensorTask	osPriorityBelowNormal	readSonarTask	osPriorityAboveNormal
pidTask	osPriorityHigh	telemetryLoggerTask	osPriorityLow
lightTask	osPriorityAboveNormal		

Tabella 8.3: Task e priorità assegnate sulle due board

### 8.7.3 Gestione delle deadline

La gestione delle deadline nel sistema è stata realizzata tramite un meccanismo dedicato, denominato *Deadline Watchdog*, progettato per monitorare il rispetto dei vincoli temporali dei task critici e rilevare eventuali violazioni di deadline in modo affidabile e deterministico.

Il *Deadline Watchdog* è implementato come un modulo software che si appoggia a un timer hardware dello STM32 configurato in modalità base con interrupt. A ciascun watchdog è associato un insieme di task monitorati, rappresentato tramite una maschera di bit (*target\_mask*), che identifica i task che devono completare correttamente la propria esecuzione entro l'intervallo temporale assegnato.

All'inizializzazione, il watchdog viene configurato associandogli il timer hardware, la maschera dei task da monitorare e una funzione di callback da invocare in caso di violazione della deadline. Contestualmente, il contatore del timer viene azzerato e il timer viene avviato con interrupt abilitato, definendo implicitamente la finestra temporale entro cui tutti i task devono segnalare il completamento.

Ogni task monitorato, al termine della propria esecuzione periodica, notifica il watchdog invocando una funzione di segnalazione dedicata. Tale notifica aggiorna una maschera di stato interna (*current\_mask*), indicando che il task ha rispettato la propria deadline. L'aggiornamento della maschera avviene all'interno di una sezione critica, così da garantire la coerenza dei dati anche in presenza di preemption da parte dello scheduler di FreeRTOS.

Quando tutte le notifiche attese sono state ricevute, ossia quando la maschera corrente coincide con la maschera target, il watchdog considera il ciclo temporale completato correttamente. In questo caso, la maschera viene azzerata e il contatore del timer viene resettato, avviando una nuova finestra di monitoraggio.

Nel caso in cui il timer hardware raggiunga la scadenza senza che tutti i task abbiano segnalato il completamento, viene generato un interrupt. In tale condizione, il watchdog arresta il timer e invoca la funzione di callback associata, che implementa la procedura di gestione della violazione di deadline. Questa procedura può includere, a seconda delle scelte progettuali, azioni di recovery, segnalazioni di errore o l'attivazione di modalità di funzionamento degradate.

Questa soluzione consente di separare in modo chiaro la logica applicativa dei task dal meccanismo di supervisione temporale, garantendo un controllo centralizzato delle deadline e mantenendo un comportamento prevedibile e coerente con i requisiti real-time del sistema.

# **Parte IV**

## **Controllo dei motori**

---

# Identificazione dei motori

---

In questo capitolo è descritta la procedura di identificazione del modello dei motori utilizzati come attuatori principali del sistema di locomozione del rover, con l'obiettivo di ottenere una rappresentazione adeguata ai fini della progettazione delle strategie di controllo. Il sistema impiega quattro motori in corrente continua con riduttore, modello **36GP-540-51**, alimentati a **12 V** e dotati di encoder incrementale integrato.

Tali attuatori costituiscono il componente fondamentale per il movimento del veicolo e la loro corretta modellazione risulta essenziale per lo sviluppo delle funzioni di controllo della velocità, di frenata e di manovra previste dalle specifiche di progetto. Le principali caratteristiche elettriche e meccaniche dei motori sono ricavate dal datasheet del produttore, disponibile al seguente indirizzo: <https://cdn.robotshop.com/media/2/2dm/rb-2dm-02/pdf/36gp540-51-en.pdf>

La procedura di identificazione tiene inoltre conto della configurazione del sistema di misura della velocità, basata su un encoder incrementale montato direttamente sull'albero del motore, a monte del riduttore. Di conseguenza, la misura fornita dall'encoder si riferisce alla rotazione dell'albero interno del motore e non a quella dell'albero di uscita del gearbox. Conoscendo il numero di impulsi per giro dell'encoder (*PPR, pulses per revolution*) e il rapporto di riduzione del riduttore, è possibile ricostruire la velocità angolare in uscita a partire dalla velocità misurata sull'albero del motore.

## 9.1 Modello dinamico del motore in corrente continua

### 9.1.1 Modello teorico

Il comportamento del motore in corrente continua può essere descritto mediante un modello dinamico ottenuto a partire dalle leggi fisiche che ne governano il funzionamento, combinando la dinamica elettrica dell'armatura con la dinamica meccanica del rotore. Il modello ha lo scopo di mettere in evidenza la struttura ingresso–uscita del sistema ed è utilizzato come riferimento per la successiva fase di identificazione.

Dal punto di vista elettrico, il motore è descritto attraverso il circuito equivalente dell'armatura, dal quale, applicando le leggi di Kirchhoff, si ottiene una relazione che lega la tensione applicata ai terminali alla corrente di armatura e alla forza contro-elettromotrice generata dalla rotazione dell'albero. Quest'ultima risulta proporzionale alla velocità angolare del motore.

La dinamica meccanica è invece descritta dall'equazione di equilibrio delle coppie agenti sul rotore, che mette in relazione la coppia elettromagnetica sviluppata dal motore con l'inerzia equivalente del sistema e con gli attriti viscosi. La coppia generata risulta a sua volta proporzionale alla corrente che attraversa l'armatura, attraverso la costante elettromeccanica del motore.

Combinando le equazioni elettriche e meccaniche e passando al dominio di Laplace, è possibile ricavare una relazione ingresso–uscita che lega la tensione applicata ai terminali del motore alla velocità angolare dell'albero. Tale relazione può essere espressa, in forma compatta, mediante la funzione di trasferimento:

$$\frac{\omega(s)}{v(s)} = \frac{k}{(sL + R)(sI + b) + k^2}.$$

Il modello così ottenuto rappresenta una descrizione dinamica completa del motore in corrente continua.

### 9.1.2 Semplificazione del modello

Nella maggior parte delle applicazioni di controllo di velocità, la dinamica elettrica del motore risulta significativamente più rapida di quella meccanica. In particolare, la costante di tempo elettrica  $\frac{L}{R}$  è generalmente molto più piccola della costante di tempo meccanica associata al sistema. Per questo motivo, l'induttanza dell'armatura può essere trascurata senza compromettere la descrizione del comportamento dominante del motore nella banda di frequenze di interesse.

Trascurando l'induttanza e riordinando i termini, il modello si riduce a una funzione di trasferimento del primo ordine:

$$G(s) = \frac{\omega(s)}{v(s)} = \frac{k_m}{1 + \tau s},$$

dove  $\tau$  rappresenta la costante di tempo del motore e  $k_m$  il guadagno statico.

Questo modello semplificato verrà utilizzato come riferimento nella successiva fase di identificazione sperimentale.

## 9.2 Acquisizione dei dati

L'acquisizione dei dati costituisce una fase fondamentale nel processo di identificazione della dinamica del motore, poiché consente di osservare la risposta del sistema a un ingresso noto. In questa fase il motore viene eccitato mediante uno stimolo a gradino di tensione, applicato tramite un alimentatore da banco, mentre i segnali di interesse vengono acquisiti in modo sincrono.

L'acquisizione è stata effettuata utilizzando un oscilloscopio digitale, impiegato per registrare sia il segnale proveniente dall'encoder incrementale sia il segnale di trigger associato all'istante di applicazione dello step. In particolare:

- il segnale dell'encoder consente di ricostruire l'andamento della velocità del motore;
- il segnale di trigger permette di individuare con precisione l'istante di inizio dello stimolo.

L'oscilloscopio utilizzato è un Hantek 6074BC, un modello USB a quattro canali con banda passante di 70 MHz e frequenza di campionamento massima pari a 1 GSa/s, idoneo all'acquisizione delle forme d'onda nel dominio del tempo.

Durante l'acquisizione sono stati impostati i seguenti parametri:

- tempo di campionamento pari a  $40 \mu\text{s}$ ;
- buffer di acquisizione di 64k campioni, corrispondente a una durata complessiva di circa 2.56 s.

La scelta di tali parametri garantisce una risoluzione temporale sufficiente a descrivere la dinamica transitoria del motore e una finestra di osservazione adeguata a includere sia il transitorio sia una porzione significativa della risposta a regime, fornendo un dataset appropriato per la successiva fase di identificazione.

## 9.3 Elaborazione dei dati in MATLAB

Per elaborare i segnali acquisiti dall'oscilloscopio e ricavare la velocità del motore nel dominio del tempo è stato sviluppato uno script in *Matlab* che automatizza le principali operazioni di preprocessing.

I dati grezzi esportati dall'oscilloscopio sono inizialmente contenuti in un unico file .txt comprendente entrambi i canali acquisiti; tali dati sono stati successivamente separati in due file distinti, **ch1.txt** e **ch2.txt**, corrispondenti rispettivamente al segnale dell'encoder e al segnale di trigger. Di seguito se ne descrivono i passaggi fondamentali.

### 9.3.1 Caricamento e allineamento dei segnali

Vengono poi caricati i segnali monocolonna esportati dall'oscilloscopio: **ch1.txt** (encoder) e **ch2.txt** (trigger reale). Inoltre, si imposta il tempo di campionamento  $T_s$  utilizzato in acquisizione.

```

1 % sampling time
2 sampling_time = 0.00004;
3
4 % === Import dei file monocolonna ===
5 encoder = readmatrix('ch1.txt');           % ch1 = encoder
6 trigger = readmatrix('ch2.txt');           % ch2 = trigger reale

```

Usando la funzione `detect_start_from_trigger`, si individua l'indice del fronte di step nel segnale di trigger (`idx_start`) e l'istante temporale corrispondente. Tale informazione viene usata per riallineare i dati imponendo che  $t = 0$  coincida con l'applicazione effettiva del gradino.

```

1 % === Stima dello starting time dal trigger ===
2 [t_start, idx_start] = ...
3 detect_start_from_trigger(trigger, sampling_time);

```

A questo punto si eliminano tutti i campioni antecedenti al fronte, ottenendo segnali che partono dal gradino.

```

1 % === Elimina tutto cio che e prima di t_start ===
2 encoder = encoder(idx_start:end);
3 trigger = trigger(idx_start:end);

```

Infine si costruisce il vettore temporale associato ai campioni rimanenti, usando  $t[k] = kT_s$  con  $k = 0, \dots, N - 1$ .

```

1 % === Costruzione vettore temporale ===
2 N = length(encoder);
3 k = (0:N-1)';
4 time = (k * sampling_time);
5
6 ch1 = encoder;

```

Di seguito si mostra un plot dei segnali grezzi:

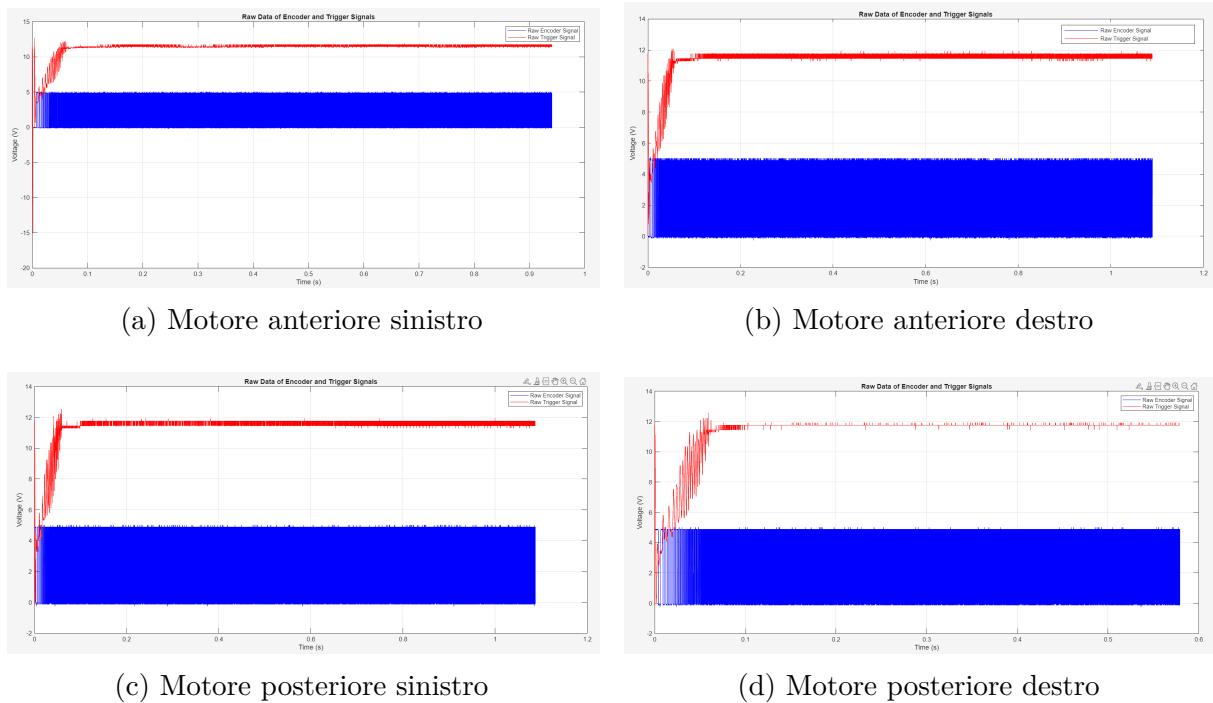


Figura 9.1: Plot delle risposte grezze in velocità dei quattro motori del rover

**Funzione di supporto (rilevamento dello step)** La funzione seguente stima l'istante di applicazione del gradino a partire dal segnale di trigger. Dopo aver ignorato una breve finestra iniziale per ridurre falsi rilevamenti, calcola la derivata discreta del trigger e identifica il primo campione in cui essa supera una soglia prefissata. Restituisce l'indice del fronte e il corrispondente istante temporale.

```

1 function [t_start, idx] = detect_start_from_trigger(trigger, Ts)
2 % Ignora i primi 0.01 s
3 ignore_time = 0.01;
4 ignore_samples = round(ignore_time / Ts);
5 % Derivata discreta del trigger
6 dtrig = diff(trigger);
7 % Soglia sul salto
8 threshold = 1.0;
9 % Cerca il primo fronte dopo ignore_samples
10 idx = find(abs(dtrig) > threshold & ...
11             (1:length(dtrig))' > ignore_samples, ...
12             1, 'first');
13 if isempty(idx)
14     error('detect_start_from_trigger:NoEdge', ...
15           'Nessun fronte rilevato.');
16 end
17 % Tempo corrispondente
18 t_start = idx * Ts;
19 end

```

### 9.3.2 Costruzione dei segnali normalizzati

Dopo l'allineamento temporale, lo script costruisce un segnale di gradino ideale (`trigger_step`) di ampiezza costante (ad esempio 12 V), utilizzato come riferimento pulito per l'analisi e la sovrapposizione grafica. Questo segnale rappresenta l'ingresso nominale applicato al sistema.

```
1 %% === Costruzione del gradino ideale ===
2 trigger_step = zeros(N,1);
3 trigger_step(1:end) = 12;
```

Il segnale dell'encoder viene convertito da tensione analogica (0–5 V) a forma d'onda digitale. Per farlo si applica una soglia a metà scala (2.5 V):

- i campioni superiori alla soglia vengono portati a 5 V;
- i campioni inferiori alla soglia vengono portati a 0 V.

Si ottiene così una sequenza di impulsi a onda quadra (`ch1_norm`), adatta alla ricerca dei fronti di salita.

```
1 ch2_norm = trigger_step;
2
3 %% === Normalizzazione encoder ===
4 threshold_encoder = 2.5;
5 ch1_norm = ch1 > threshold_encoder;
6 ch1_norm = ch1_norm * 5;
```

Di seguito si mostrano i plot dei segnali opportunatamente normalizzati:

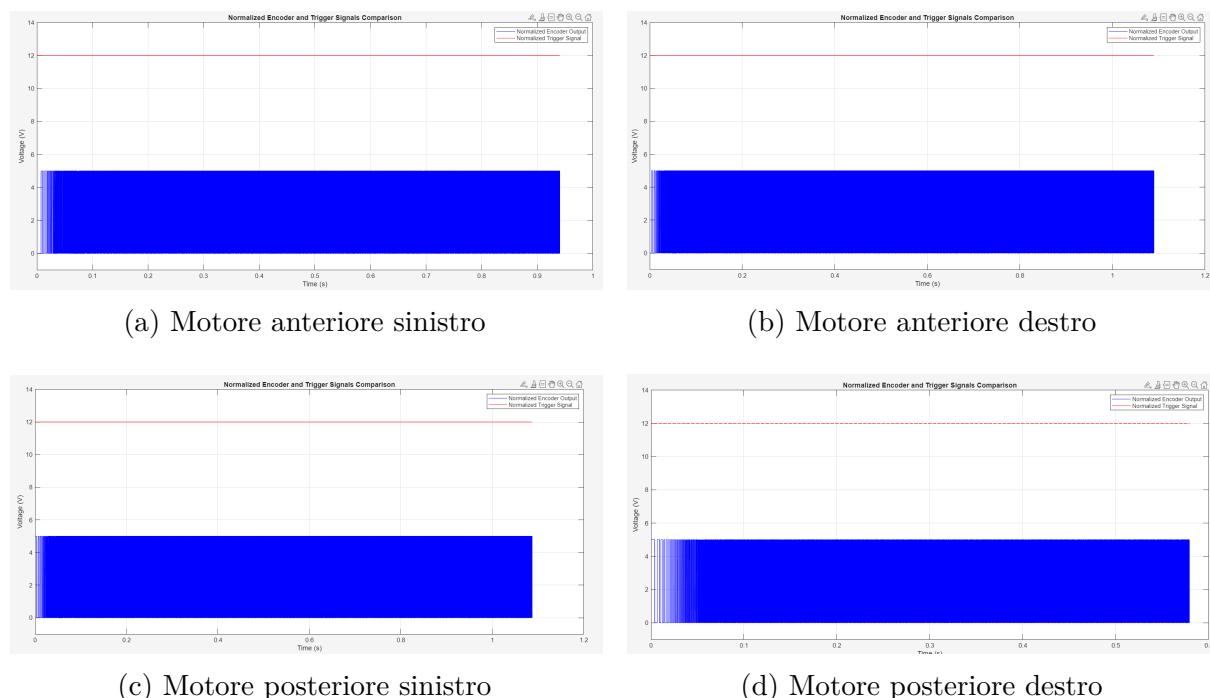


Figura 9.2: Risposte normalizzate in velocità dei quattro motori del rover

### 9.3.3 Interpolazione temporale e segnali finali

A partire dal segnale dell'encoder, il segnale analogico viene dapprima convertito in forma logica mediante l'applicazione di una soglia (`threshold_encoder`), ottenendo un vettore binario `enc_logic`. I fronti di salita vengono quindi individuati come le transizioni da 0 a 1 del segnale logico, identificate tramite la differenza discreta `diff(enc_logic) == 1`. Gli istanti temporali corrispondenti a ciascun fronte di salita sono raccolti nel vettore `rising_times`.

```

1 %% === Calcolo RPM robusto da impulsi encoder ===
2 enc_logic = ch1 > threshold_encoder;
3 rising_edges = [false; diff(enc_logic) == 1];
4 rising_times = time(rising_edges);

```

A partire dagli istanti temporali in cui si verificano i fronti di salita dell'encoder, indicati con  $t_1, t_2, \dots, t_N$ , il segnale disponibile non è una misura continua della velocità, ma una sequenza di eventi discreti nel tempo. Ciascun fronte di salita rappresenta il completamento di una frazione fissa di giro dell'albero, ma non fornisce direttamente informazioni sulla velocità istantanea in un singolo istante.

Per estrarre una stima della velocità a partire da tali eventi discreti, si considerano quindi gli intervalli di tempo tra due fronti consecutivi, definiti come

$$\Delta t_i = t_{i+1} - t_i.$$

Tale intervallo rappresenta il tempo necessario affinché l'albero percorra una quantità angolare nota, determinata dalla risoluzione dell'encoder. Assumendo che la velocità sia approssimativamente costante all'interno di ciascun intervallo  $\Delta t_i$ , è possibile stimare la frequenza degli impulsi come

$$f_i = \frac{1}{\Delta t_i}.$$

La frequenza così ottenuta non è naturalmente associabile né all'istante  $t_i$  né all'istante  $t_{i+1}$ , poiché essa è calcolata utilizzando informazioni provenienti da entrambi i fronti. Per evitare un'assegnazione temporale arbitraria e introdurre un errore di allineamento temporale, la stima di frequenza viene quindi riferita al tempo centrale dell'intervallo, definito come

$$t_i^{\text{mid}} = t_i + \frac{\Delta t_i}{2}.$$

```

1 dt = diff(rising_times); % s
2 t_mid = rising_times(1:end-1) + dt/2; %tempo associato alla misura
3 f_inst = 1 ./ dt; % Hz

```

La sequenza di frequenze istantanee  $f_i$  ottenuta a partire dagli intervalli tra impulsi può risultare affetta da valori anomali (*outlier*), che non riflettono il reale andamento della velocità del motore. Tali valori spurii possono essere causati, ad esempio, da jitter temporale nella rilevazione dei fronti o da rumore sul segnale dell'encoder. Poiché la frequenza è calcolata come l'inverso dell'intervallo temporale, anche una piccola perturbazione su  $\Delta t_i$  può tradursi in un picco isolato di grande ampiezza in  $f_i$ .

Per mitigare questi effetti e rendere la stima più robusta, la sequenza  $f_i$  viene sottoposta a una procedura automatica di individuazione e correzione degli outlier tramite la funzione `filloutliers`. In particolare, viene utilizzato un criterio basato sulla mediana mobile (`movmedian`) calcolata su una finestra di ampiezza prefissata. La mediana mobile fornisce

una stima locale dell'andamento del segnale che risulta poco sensibile a valori estremi isolati.

I campioni identificati come outlier vengono quindi sostituiti mediante interpolazione lineare tra i valori adiacenti non anomali.

La scelta della dimensione della finestra di mediana rappresenta un compromesso tra la capacità di soppressione del rumore impulsivo e la fedeltà nella rappresentazione delle variazioni rapide della velocità.

```
1 f_clean = filloutliers(f_inst, "linear", "movmedian", 9);
```

La frequenza degli impulsi dell'encoder viene quindi convertita in velocità di rotazione del motore, espressa in giri al minuto (RPM), tenendo conto del numero di impulsi per giro dell'encoder (PPR) e del rapporto di riduzione del riduttore (GEAR\_RATIO), secondo la relazione:

$$\text{RPM}(t) = \frac{f_{\text{impulsi}}(t) \cdot 60}{\text{PPR} \cdot \text{GEAR\_RATIO}}.$$

```
1 PPR = 12;
2 GEAR_RATIO = 51;
3 rpm_meas = (f_clean * 60) / (PPR * GEAR_RATIO);
```

Poiché la velocità del motore viene stimata a partire dagli impulsi dell'encoder, essa risulta definita solo in corrispondenza degli istanti associati agli intervalli tra due fronti consecutivi, ovvero sui tempi  $t_i^{\text{mid}}$ . Di conseguenza, la sequenza `rpm_meas` non è campionata in modo uniforme nel tempo e non può essere direttamente confrontata con altri segnali acquisiti a frequenza costante, né utilizzata in modo affidabile per successive operazioni di identificazione del modello. Per superare questa limitazione, viene introdotta una nuova griglia temporale uniforme  $\{t_q\}$ , definita con una frequenza di ricostruzione  $F_s_{\text{rpm}} = 500$  (sufficientemente elevata rispetto alla dinamica del sistema e molto inferiore alla frequenza di acquisizione).

L'interpolazione della velocità sulla nuova griglia temporale viene effettuata mediante la funzione `interp1` utilizzando il metodo `pchip` scelto in quanto realizza un'interpolazione cubica a tratti che preserva l'andamento locale del segnale, evitando oscillazioni non fisiche tra i campioni. Con `fillmissing` i campioni mancanti agli estremi della sequenza interpolata vengono riempiti assegnando il valore valido più vicino nel tempo.

Successivamente il segnale di riferimento `ch2_norm` viene a sua volta riportato sulla medesima griglia temporale uniforme. In questo caso viene utilizzato un criterio di mantenimento del valore precedente (`previous`), coerente con la natura a gradino del segnale di comando.

```
1 Fs_rpm = 500;
2 tq = (time(1):1/Fs_rpm:time(end)).';
3
4 rpm_i = interp1(t_mid, rpm_meas, tq, 'pchip');
5 rpm_i = fillmissing(rpm_i, 'nearest');
6
7 ch2_i = interp1(time, ch2_norm, tq, 'previous');
```

Per ridurre ulteriormente il rumore residuo e ottenere una curva di velocità più regolare, la velocità interpolata viene sottoposta a una fase di smoothing tramite un filtro.

```
1 rpm_s = smoothdata(rpm_i, "sgolay", 31);
```

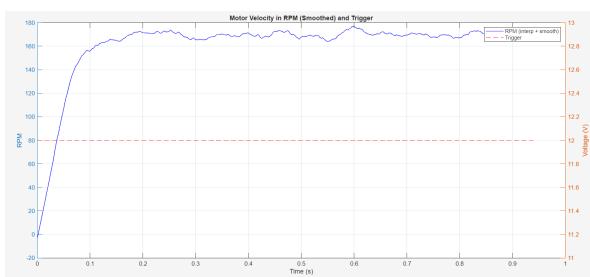
### 9.3.4 Visualizzazione dei risultati

Infine, viene generata una figura che mostra l'andamento temporale della velocità del motore in RPM, ottenuta tramite interpolazione e filtraggio, confrontata con il segnale di comando a gradino.

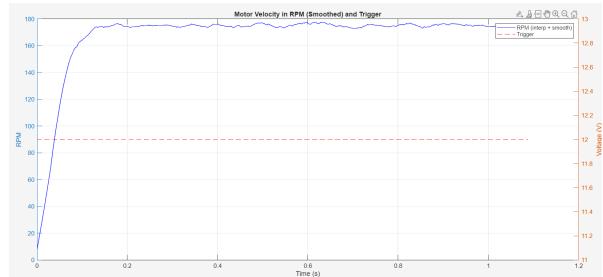
```

1 %% === Plot RPM filtrati + trigger ===
2 figure;
3 yyaxis left;
4 plot(tq, rpm_s, 'b', 'DisplayName', 'RPM (interp + smooth)');
5 ylabel('RPM');
6
7 yyaxis right;
8 plot(tq, ch2_i, 'r--', 'DisplayName', 'Trigger');
9 ylabel('Voltage (V)');
10
11 title('Motor Velocity in RPM (Smoothed) and Trigger');
12 xlabel('Time (s)');
13 legend;
14 grid on;

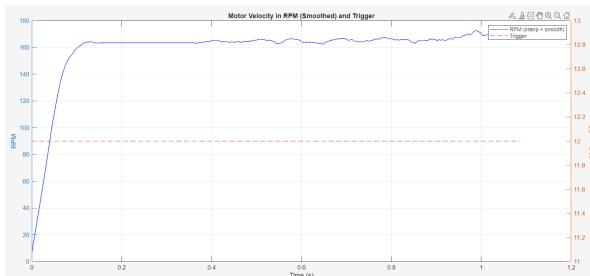
```



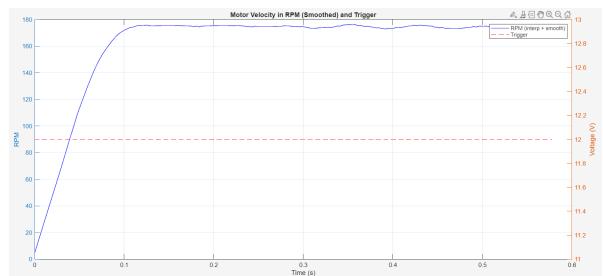
(a) Motore anteriore sinistro



(b) Motore anteriore destro



(c) Motore posteriore sinistro



(d) Motore posteriore destro

Figura 9.3: Grafico Rpm-tensione per i quattro motori del rover

## 9.4 System Identification Tool di MATLAB

Una volta ricostruita la velocità angolare del motore e sincronizzati i segnali di ingresso e uscita tramite lo script *Matlab*, la fase successiva consiste nell'importare tali dati all'interno del *System Identification Tool* (SIT) per eseguire la stima del modello dinamico. Il SIT fornisce un ambiente grafico altamente strutturato che consente l'identificazione di un modello dinamico a partire da dati sperimentali. La procedura adottata è descritta di seguito dettagliatamente.

### 9.4.1 Apertura dello strumento

Lo strumento viene avviato da *Matlab* tramite il comando:

```
>> systemIdentification
```

All'avvio compare un'unica finestra che costituisce il cuore del processo di identificazione. Essa è divisa logicamente in tre zone:

- *Data Views* (a sinistra), dove vengono visualizzati i dataset importati;
- *Working Data* (al centro), che rappresenta il dataset attivamente utilizzato nelle operazioni di preprocessing e stima;
- *Model Views* (a destra), dove compaiono i modelli stimati e i relativi grafici di validazione.

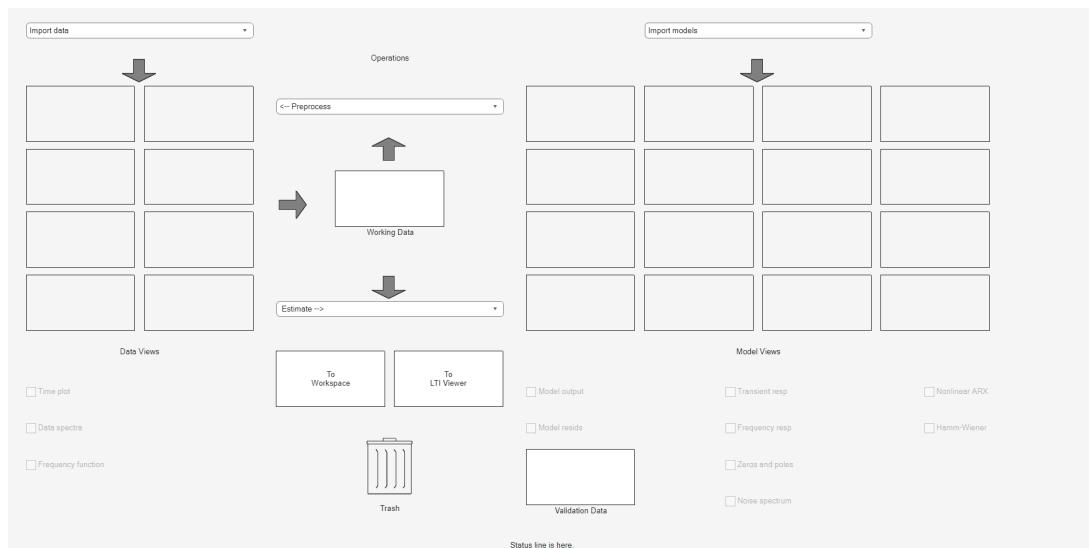


Figura 9.4: Schermata di home del System Identification Tool

### 9.4.2 Importazione dei dati nel dominio del tempo

Il primo passo consiste nell'importare i dati sperimentali elaborati in precedenza. Attraverso il menu **Import data** si seleziona la voce **Time domain data....**

Viene così aperto un pannello nel quale si specificano:

- l'ingresso  $u(t)$ , rappresentato dal gradino ideale a 12 V ricostruito nello script (`ch2_i`);

- l'uscita  $y(t)$ , cioè la velocità in RPM ottenuta dall'elaborazione dei fronti dell'encoder (`rpm_s`);
- il tempo di campionamento  $T_s = 40 \mu s$ .

Confermando l'importazione, il dataset viene automaticamente aggiunto alla sezione *Data Views* e contemporaneamente impostato come *Working Data*. Questo permette di utilizzare immediatamente i segnali per la fase successiva.

### 9.4.3 Verifica visiva e preprocessing

Prima di procedere alla stima del modello, il SIT consente di ispezionare il dataset tramite grafici temporali e strumenti di verifica dei segnali. In generale sarebbe possibile:

- rimuovere offset indesiderati;
- applicare filtri passa-basso;
- ritagliare intervalli inutili o distorti;
- normalizzare o centrare i dati.

Nel nostro caso, tali operazioni non sono state necessarie poiché l'intero processo di pulizia e sincronizzazione dei segnali era già stato preventivamente via script Matlab.

### 9.4.4 Selezione della struttura del modello

Per accedere alla fase di identificazione si utilizza il menu **Estimate**. Tra le diverse opzioni disponibili, è stata scelta la voce **Transfer Function Models**....

Aprendo la finestra dedicata alla stima, vengono richiesti alcuni parametri relativi alla struttura del modello. In particolare:

- **Model name:** `tf_motore`;
- **Number of poles:** 1;
- **Number of zeros:** 0;
- **Domain:** modello continuo (continuous-time);
- **Input delay:** fissato a 0.

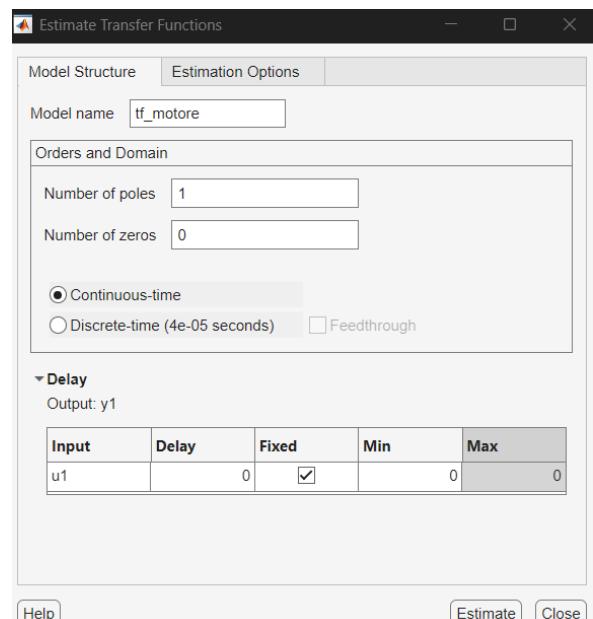


Figura 9.5: Configurazione della funzione di trasferimento

Questi parametri corrispondono alla funzione di trasferimento teorica di un motore DC:

$$G(s) = \frac{K}{\tau s + 1},$$

dove  $K$  rappresenta il guadagno statico e  $\tau$  la costante di tempo del sistema.

La struttura del modello selezionata è coerente con la modellazione introdotta in precedenza e permette di concentrare l'identificazione sulla stima dei parametri dinamici principali del sistema.

#### 9.4.5 Stima del modello

Una volta specificata la struttura, tramite il pulsante **Estimate** viene avviata la procedura di identificazione che, basandosi sull'algoritmo di *Prediction Error Minimization* (PEM), cerca i valori ottimali dei parametri  $K$  e  $\tau$  tali da minimizzare la differenza tra la risposta del modello e l'uscita reale misurata.

Al termine della stima, il modello ottenuto viene mostrato all'interno della sezione *Model Views*.

#### 9.4.6 Validazione del modello

Una parte fondamentale dell'identificazione consiste nella verifica della qualità del modello stimato. Il SIT permette di eseguire tale validazione in modo immediato, sovrapponendo la risposta del modello all'uscita reale. Nel caso dei nostri dataset, il modello del primo ordine mostrava un'ottima capacità di riprodurre il comportamento dinamico del motore, con un fit elevato, segno che la struttura scelta era adeguata. A titolo esemplificativo, in Figura 9.6 è riportato il risultato della validazione del modello stimato per uno dei quattro motori del rover, in particolare il motore anteriore destro.

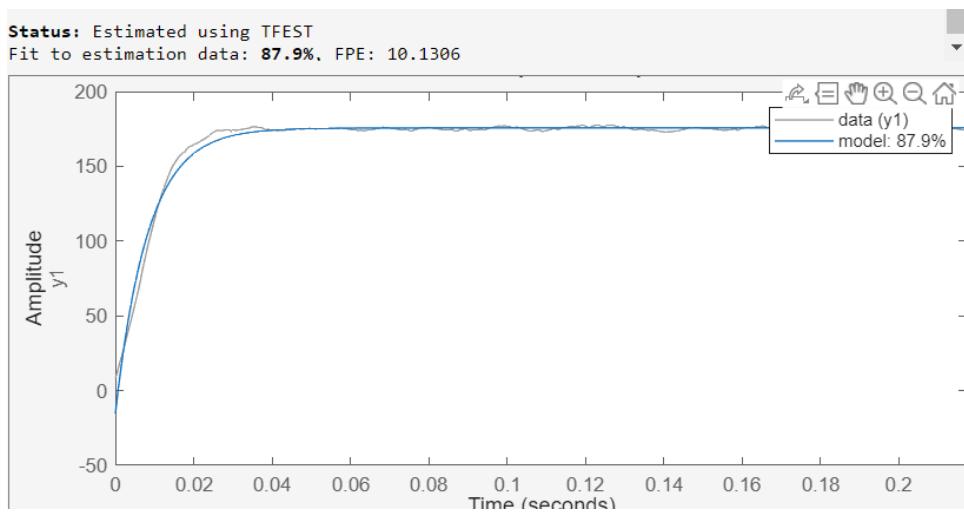


Figura 9.6: Validazione visuale del modello

#### 9.4.7 Esportazione del modello e utilizzo finale

Infine, tramite il pulsante **To Workspace**, la funzione di trasferimento stimata viene esportata come oggetto Matlab. Questo permette di utilizzarla nella fase successiva di progettazione del controllore PID.

## 9.5 Modelli Identificati per i Motori del Rover

A seguito della procedura di identificazione descritta nelle sezioni precedenti, sono state stimate le funzioni di trasferimento continue del primo ordine per ciascuno dei quattro motori del rover. Le stime sono state ottenute utilizzando l'algoritmo TFEST sui dati sperimentali acquisiti e preprocessati tramite lo script Matlab.

Per ogni motore si riporta di seguito la funzione di trasferimento identificata, insieme ai principali indici di qualità della stima (FPE<sup>1</sup> e MSE<sup>2</sup>).

Motore	G(s)	Fit (%)	FPE	MSE
Anteriore sinistro	$\frac{14.203}{1 + 0.008475 s}$	87.6	12.8	12.64
Anteriore destro	$\frac{14.639}{1 + 0.008299 s}$	87.9	10.13	10.02
Posteriore sinistro	$\frac{13.817}{1 + 0.007936 s}$	85.62	12.32	12.18
Posteriore destro	$\frac{14.675}{1 + 0.008019 s}$	85.4	16.45	15.91

Tabella 9.1: Modelli dinamici identificati e indici di qualità per i motori del rover

<sup>1</sup>La *Final Prediction Error* (FPE) è un indice di qualità della stima che fornisce una misura dell'errore di predizione atteso su dati non utilizzati durante l'identificazione, penalizzando modelli eccessivamente complessi. Valori più bassi di FPE indicano una migliore capacità di generalizzazione del modello.

<sup>2</sup>La *Mean Squared Error* (MSE) rappresenta invece il valore medio del quadrato dell'errore tra l'uscita misurata e quella stimata dal modello sui dati di identificazione; anch'essa assume valori minori per modelli più accurati.

---

# Progettazione dei controllori per i motori

---

In questo capitolo viene presentato l'approccio adottato per la progettazione dei controllori dei motori del rover. Dopo una breve introduzione ai principi del controllo in retroazione, viene descritta la struttura dei controllori utilizzati e motivate le principali scelte progettuali effettuate. Sulla base dei modelli dinamici dei motori, vengono infine illustrati i criteri adottati per la sintesi e la valutazione delle prestazioni del sistema di controllo.

## 10.1 Controllo in retroazione e regolazione PID

Il passo successivo consiste nella progettazione di una strategia di controllo in grado di regolare la velocità dell'albero motore in presenza di variazioni del riferimento e di disturbi esterni. Nel contesto del progetto, tale obiettivo viene perseguito mediante l'impiego di un controllo in retroazione, che consente di confrontare continuamente il valore desiderato della velocità con quello misurato mediante gli encoder per ciascun motore.

### 10.1.1 Controllo in retroazione

Nel controllo in retroazione, detto anche controllo a ciclo chiuso, l'uscita del sistema viene misurata e confrontata con un valore di riferimento. La differenza tra il riferimento e l'uscita, definita errore di controllo, viene utilizzata per generare il segnale di comando applicato al sistema.

Indicando con  $\omega_r(t)$  la velocità di riferimento, con  $\omega(t)$  la velocità misurata e con  $e(t)$  l'errore, si ha:

$$e(t) = \omega_r(t) - \omega(t).$$

Il segnale di controllo  $u(t)$  viene quindi ottenuto elaborando l'errore tramite un opportuno controllore, ed è applicato all'ingresso del motore sotto forma di tensione.

L'adozione della retroazione permette di ridurre la sensibilità del sistema alle variazioni dei parametri del motore, agli errori di modellazione e ai disturbi esterni, migliorando l'accuratezza e la robustezza della regolazione della velocità.

### 10.1.2 Struttura del controllore PID

Tra le diverse tipologie di controllori utilizzabili nei sistemi a retroazione, il controllore PID (Proporzionale–Integrale–Derivativo) rappresenta una soluzione consolidata per la regolazione di sistemi dinamici continui.

Il controllore PID genera il segnale di comando a partire dall'errore secondo la legge:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt},$$

dove  $K_p$ ,  $K_i$  e  $K_d$  sono rispettivamente il guadagno proporzionale, integrativo e derivativo. Il termine proporzionale fornisce un'azione immediata in risposta all'errore, influenzando la rapidità della risposta del sistema. Il termine integrativo accumula l'errore nel tempo e consente di eliminare l'errore a regime. Il termine derivativo, infine, introduce un'azione anticipativa basata sulla variazione dell'errore, contribuendo allo smorzamento delle oscillazioni e al miglioramento della stabilità dinamica (beneficio sul transitorio).

## 10.2 Progettazione del Controllore PID tramite Sisotool

Ricostruito il modello dinamico del motore, la fase successiva consiste nella progettazione del controllore che dovrà regolare la velocità del motore in risposta al comando di riferimento. Per la progettazione è stato utilizzato lo strumento grafico *Sisotool* di *Matlab*, che permette di interagire direttamente con il modello del sistema e con i luoghi geometrici delle radici, i diagrammi di Bode e la risposta al gradino.

### 10.2.1 Scelta del controllore: perché un PI e non un PID

Prima di procedere alla sintesi tramite *Sisotool*, è stata valutata la tipologia di controllore più adatta alla regolazione della velocità del motore. Sebbene il controllore PID sia una scelta generale per molti sistemi dinamici, nel caso specifico della regolazione di velocità di un motore in corrente continua il termine derivativo non è necessario e può risultare addirittura controproducente. Le motivazioni principali sono:

- **La dinamica del motore è essenzialmente di primo ordine.** Il modello identificato presenta un solo polo dominante: ciò rende sufficiente l'azione proporzionale per controllare la rapidità della risposta e l'azione integrativa per eliminare l'errore a regime. L'azione derivativa non porta benefici significativi in termini di prestazioni.
- **Il segnale dell'encoder è un'onda quadra con fronti molto ripidi.** Il calcolo del termine derivativo amplificherebbe il rumore e i disturbi ad alta frequenza generati dai fronti dell'encoder, introducendo instabilità o oscillazioni indesiderate nel comando.
- **Il controllo della velocità richiede robustezza, non reattività estrema.** Il derivativo è utile nei sistemi in cui serve anticipare rapide variazioni della dinamica. Nel caso della velocità del motore, invece, l'obiettivo è mantenere un regime stabile e filtrare rumore e fluttuazioni, cosa che il PI garantisce in modo sufficiente.

Per queste ragioni la nostra scelta è ricaduta su un controllore PI, che rappresenta la soluzione più robusta, semplice ed efficace per la regolazione della velocità del motore.

### 10.2.2 Apertura di Sisotool e configurazione dello strumento

Richiamando il comando:

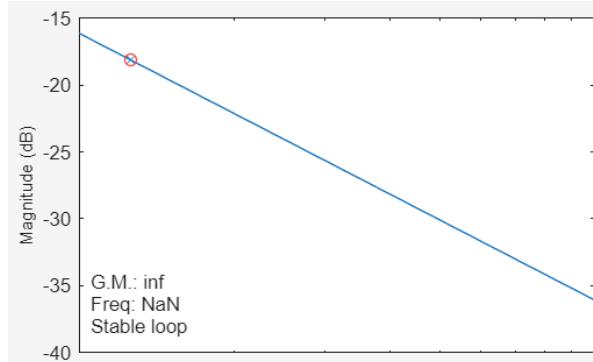
```
>> sisotool(G)
```

dove *G* rappresenta la funzione di trasferimento identificata del motore, *Sisotool* apre l'interfaccia grafica caricando automaticamente il modello dell'impianto in anello aperto. In questo modo l'intero sistema è già predisposto per l'analisi interattiva. Una volta caricato il modello dell'impianto, tramite i metodi messi a disposizione da *Sisotool* è possibile modificare la struttura del controllore aggiungendo in modo interattivo i blocchi necessari. In particolare, per implementare un controllore di tipo PI viene inserito un polo nell'origine (che rappresenta l'azione integrativa) e uno zero reale, posizionato opportunamente per migliorare la rapidità della risposta e garantire un adeguato smorzamento. Lo strumento

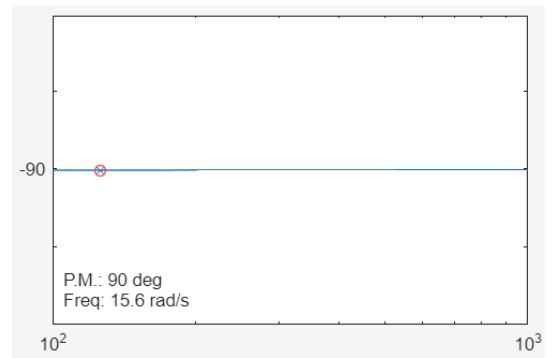
aggiorna in tempo reale il luogo delle radici, il diagramma di Bode e la risposta al gradino, permettendo di verificare immediatamente gli effetti delle modifiche introdotte.

### 10.2.3 Criteri adottati per la progettazione dei controllori

Nella progettazione del controllore è stato necessario scegliere una banda a ciclo chiuso che fosse compatibile sia con le prestazioni desiderate sia con i vincoli fisici del sistema. La banda non può essere scelta arbitrariamente alta: aumentare la velocità di risposta richiede una frequenza di attraversamento maggiore, ma questo riduce i margini di stabilità e rende il sistema più sensibile al rumore e alle variazioni dei parametri del modello. Al contrario, una banda troppo bassa genera un sistema eccessivamente lento, incapace di seguire variazioni anche moderate del riferimento. Per questo motivo, la banda è stata selezionata in modo da ottenere un compromesso tra rapidità e robustezza, garantendo una risposta sufficientemente veloce ma evitando instabilità o sovraelongazioni indesiderate. Un'altra cosa fondamentale è stato assicurarsi che in corrispondenza della frequenza di attraversamento il sistema presentasse un margine di fase adeguato. Un margine di fase elevato è fondamentale quando il controllore deve essere successivamente implementato in forma discretizzata: l'atto stesso della discretizzazione introduce un ritardo equivalente dovuto al tempo di campionamento. Se il margine di fase in continuo fosse troppo ridotto, questo porterebbe il sistema in prossimità della perdita di stabilità una volta implementato su microcontrollore. A titolo di esempio, si riportano di seguito i diagrammi di Bode della funzione di trasferimento ad anello aperto del motore anteriore sinistro. Dalla Figura si osserva che l'anello aperto presenta un margine di fase pari a circa  $PM \approx 90^\circ$  ad una frequenza di attraversamento di 15.6 rad/s.



(a) Grafico di Bode per modulo anello aperto di anteriore sinistro



(b) Grafico di Bode per fase anello aperto di anteriore sinistro

La banda del sistema, che in prima approssimazione può essere ricondotta alla frequenza di attraversamento dell'anello aperto, può essere valutata in modo più accurato analizzando il diagramma di Bode della funzione di trasferimento a ciclo chiuso, ottenuta a partire dalla funzione di trasferimento ad anello aperto  $F(s)$  e dalla funzione di retroazione  $H(s)$ , come

$$W(s) = \frac{F(s)}{1 + F(s)H(s)}.$$

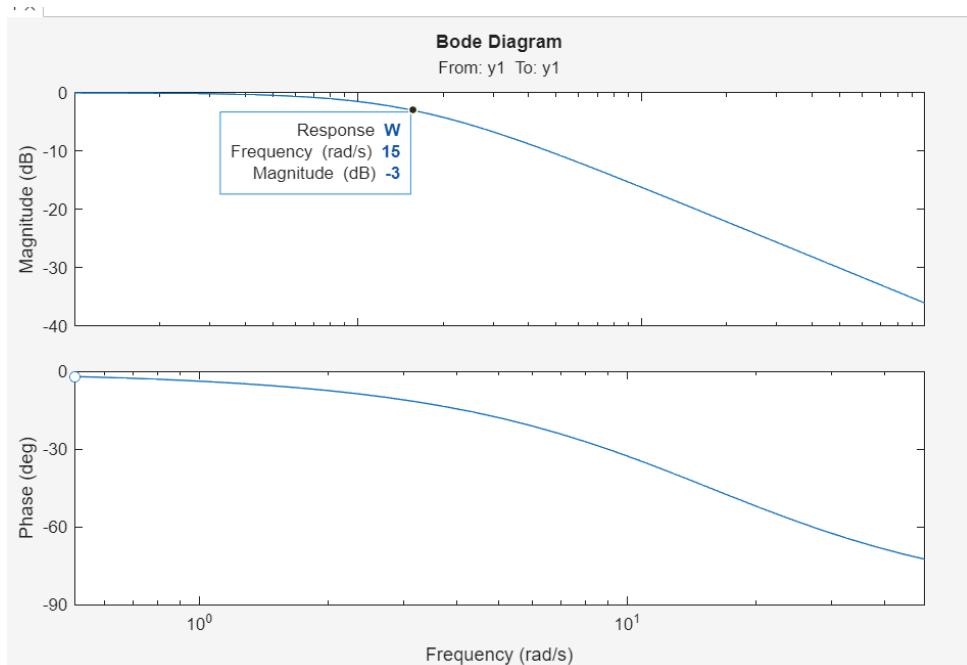


Figura 10.2: Grafico di Bode della funzione di trasferimento a ciclo chiuso per anteriore sinistro

Inoltre nella progettazione si è scelto di imporre, come requisito comune, un tempo di assestamento equivalente per tutti e quattro i controllori dei motori. Tale scelta è stata adottata per superare le differenze parametriche dei singoli modelli, con l'obiettivo di ottenere una dinamica globale coerente. Di seguito si mostrano le 4 risposte a gradino:

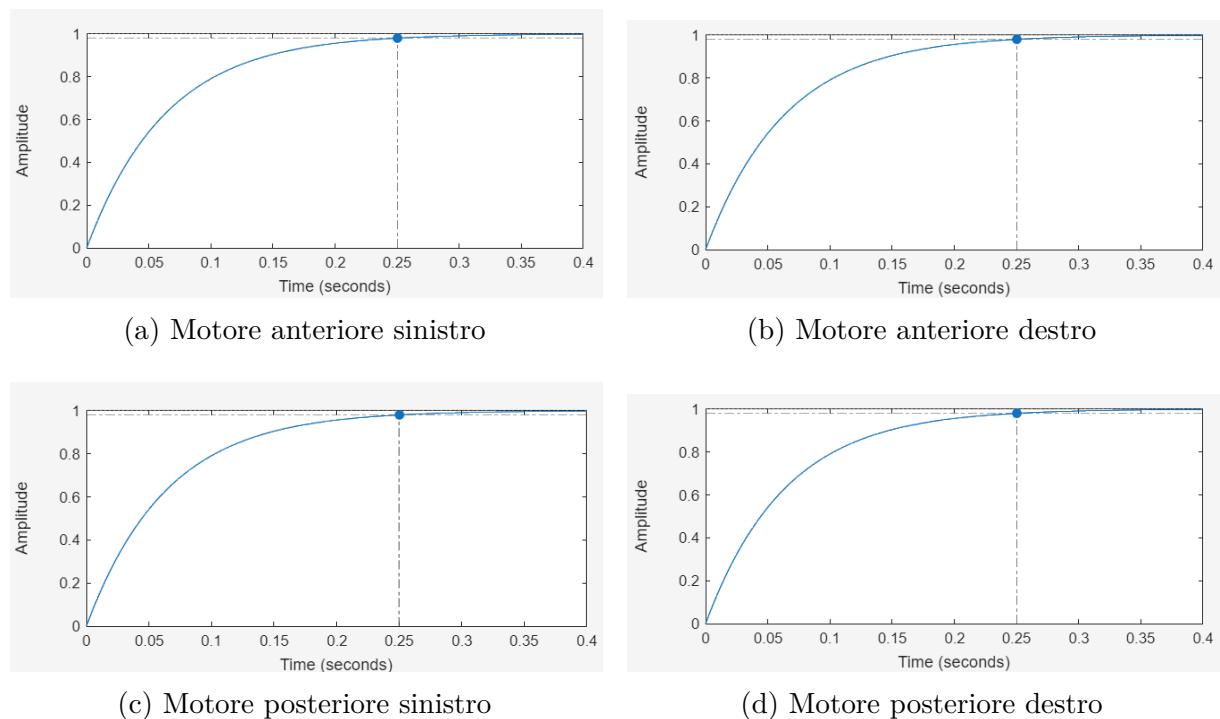


Figura 10.3: Risposta al gradino a ciclo chiuso dei quattro motori

### 10.2.4 Controllori PI nel continuo

A valle della procedura descritta nelle sezioni precedenti, sono stati progettati quattro controllori di tipo PI, uno per ciascun motore del rover.

Tutti i controllori presentano una struttura del tipo:

$$C(s) = K \frac{s + z}{s},$$

caratterizzato sia da un polo nell'origine (il quale realizza l'azione integrativa) che da uno zero reale. In Tabella 10.1 sono riportate le funzioni di trasferimento continue dei controllori PI progettati per ciascun motore.

Motore	Controllore PI continuo $C(s)$
Anteriore sinistro	$0.009319 \frac{s + 118}{s}$
Anteriore destro	$0.0088585 \frac{s + 120.5}{s}$
Posteriore sinistro	$0.0089885 \frac{s + 126}{s}$
Posteriore destro	$0.0085428 \frac{s + 124.7}{s}$

Tabella 10.1: Controllori PI continui progettati per i quattro motori del rover

### 10.2.5 Scelta del tempo di campionamento

Il controllore progettato nelle sezioni precedenti è stato inizialmente descritto e sintetizzato nel dominio continuo, facendo riferimento a modelli dinamici continui del motore e a tecniche di progetto basate su funzioni di trasferimento. Tuttavia, ai fini dell'implementazione reale del sistema di controllo, tale descrizione non è direttamente utilizzabile. In particolare, l'implementazione pratica del controllore avviene su un microcontrollore, il quale opera su segnali discreti nel tempo e con precisione finita. Ciò rende necessario introdurre un processo di digitalizzazione, che comprende la conversione analogico-digitale dei segnali misurati, l'elaborazione numerica dell'algoritmo di controllo e la successiva conversione digitale-analogica del comando applicato all'attuatore. All'interno di questo processo, il funzionamento del controllore è scandito da una sequenza di istanti temporali equispaziati, separati da un intervallo costante detto *tempo di campionamento*  $T_s$ . In corrispondenza di ciascun istante di campionamento, il microcontrollore acquisisce i segnali di ingresso, calcola il comando di controllo discretizzato e aggiorna il segnale applicato all'attuatore, che viene mantenuto costante fino all'istante successivo secondo una tecnica di *mantenimento a ordine zero* (*Zero-Order Hold*, ZOH). La scelta del tempo di campionamento influenza direttamente sul comportamento del sistema di controllo. Un campionamento troppo lento introduce perdita di informazione e ritardi equivalenti che possono degradare le prestazioni o compromettere la stabilità del sistema; viceversa, un campionamento eccessivamente rapido non garantisce un avvicinamento illimitato al comportamento continuo, poiché la risoluzione finita della computazione digitale può rendere le variazioni tra campioni non significative, con conseguente perdita di informazione utile. Sulla base di

considerazioni empiriche ampiamente adottate nei sistemi di controllo digitale, il tempo di campionamento può essere scelto in relazione alla dinamica dominante del sistema o alle prestazioni richieste a ciclo chiuso. Nel nostro caso, la scelta del tempo di campionamento è stata effettuata facendo riferimento diretto alle prestazioni desiderate del sistema a ciclo chiuso. Indicando con  $T_{ss}$  il tempo di assestamento richiesto e con  $\omega_n$  la pulsazione dominante associata alla dinamica del sistema, sono stati considerati i criteri

$$T_s < \frac{T_{ss}}{10} \quad \text{oppure} \quad \omega_s \geq \alpha \omega_n,$$

dove  $\omega_s = \frac{2\pi}{T_s}$  è la pulsazione di campionamento e  $\alpha$  è un fattore di sicurezza tipicamente compreso tra 5 e 20.

Sulla base di tali considerazioni, considerando un tempo di assestamento richiesto  $T_{ss} = 0,25$  s, si è scelto un tempo di campionamento tale da risultare significativamente più piccolo di  $T_{ss}/10$ , in modo da garantire una discretizzazione adeguata della dinamica a ciclo chiuso. In particolare, il tempo di campionamento è stato fissato a:

$$T_s = 5 \text{ ms},$$

### 10.2.6 Implementazione digitale

Una volta fissato il tempo di campionamento, il controllore progettato nel dominio continuo viene tradotto in un algoritmo di controllo discreto mediante la determinazione dei parametri del PI discreto a partire dal progetto continuo.

L'azione proporzionale non richiede una vera e propria discretizzazione, poiché consiste in una moltiplicazione istantanea dell'errore. Al contrario, l'azione integrativa, che implica un'operazione di integrazione nel tempo continuo, deve essere approssimata mediante un operatore discreto.

Nel presente lavoro, la discretizzazione è stata effettuata assumendo un mantenimento a ordine zero dei segnali (*Zero-Order Hold*) e approssimando l'integrale mediante una somma discreta. In particolare, l'integratore continuo

$$\int_0^t e(\tau) d\tau$$

viene approssimato, in forma incrementale, come

$$I[k] = I[k - 1] + T_s e[k],$$

dove  $e[k]$  rappresenta l'errore campionato all'istante  $kT_s$ . Di conseguenza, il controllore PI discreto assume la forma

$$u[k] = K_p e[k] + K_i T_s \sum_{i=0}^k e[i].$$

In questa formulazione, i parametri del controllore discreto risultano direttamente derivati da quelli continui. In particolare, mentre il guadagno proporzionale rimane invariato,

$$K_p^{(d)} = K_p,$$

il guadagno integrativo discreto incorpora esplicitamente il tempo di campionamento:

$$K_i^{(d)} = K_i T_s.$$

Ciò evidenzia come la scelta del tempo di campionamento influisca direttamente sull'intensità dell'azione integrativa: a parità di parametri continui, un valore di  $T_s$  più elevato produce un incremento più marcato dell'azione integrativa tra campioni successivi.

### 10.2.7 Ritardo introdotto

L'implementazione pratica del controllore su microcontrollore introduce un ulteriore aspetto che deve essere preso in considerazione: il *ritardo computazionale*. Dal punto di vista dell'analisi in frequenza, questi effetti si manifestano principalmente come una *perdita di fase* aggiuntiva, particolarmente critica in prossimità della frequenza di attraversamento dell'anello aperto. Tale perdita di fase dipende dalla frequenza  $e$ , valutata alla pulsazione corrispondente alla banda  $\omega_f$ , può essere approssimata come

$$\varphi_d(\omega_f) \approx -\omega_f \frac{T_s}{2} \frac{180}{\pi}.$$

Per un tempo di campionamento  $T_s = 5$  ms, la riduzione di fase alla banda passante  $\omega_f = 15$  rad/s può essere approssimata come

$$\Delta\varphi \approx -\omega_f \frac{T_s}{2} \frac{180}{\pi} \approx -2.14^\circ.$$

Tale perdita di fase risulta ampiamente trascurabile rispetto al margine di fase disponibile in continuo e non altera in modo significativo le condizioni di stabilità del sistema a ciclo chiuso.

### 10.2.8 Temporizzazione dell'algoritmo

L'algoritmo di controllo digitale viene eseguito secondo la seguente sequenza operativa:

1. Acquisizione dei segnali di uscita dell'impianto mediante conversione analogico-digitale;
2. Calcolo del segnale di comando sulla base dell'algoritmo di controllo discretizzato;
3. Invio del comando all'attuatore mediante conversione digitale-analogica.

Indicando con  $t(k)$  l'istante di campionamento del segnale in ingresso al controllore, il comando calcolato viene applicato all'attuatore all'istante  $t(k) + h$ . Affinché il sistema operi correttamente in modo sincrono, è necessario che tale ritardo rimanga inferiore al tempo di campionamento, ossia  $h < T_s$ . Nel presente progetto, il valore scelto di  $T_s = 5$  ms risulta sufficientemente elevato da rendere il ritardo computazionale trascurabile rispetto alla dinamica del sistema a ciclo chiuso, evitando effetti apprezzabili sui margini di stabilità e sulla robustezza complessiva del controllo.

### 10.2.9 Problematiche di implementazione

Nel passaggio dalla progettazione teorica del controllore alla sua implementazione su un sistema reale è necessario tenere conto di una serie di fenomeni che non sono descritti dal modello utilizzato in fase di progetto. In particolare, la presenza di vincoli fisici sugli attuatori e l'implementazione pratica del controllo introducono effetti non ideali che possono compromettere le prestazioni attese del sistema.

## Saturazione

Nel sistema in esame, il comando di controllo calcolato dal regolatore viene applicato al motore sotto forma di tensione (o di un segnale equivalente), che è intrinsecamente limitata da vincoli fisici e di progetto. Quando il comando richiesto dal controllore supera tali limiti, l'attuatore entra in saturazione e il segnale effettivamente applicato differisce da quello teoricamente calcolato dal regolatore.

### Soluzione

Per tenere conto di tale aspetto, l'algoritmo di controllo deve distinguere esplicitamente tra:

- il comando teorico non saturo  $u_{\text{unsat}}$ , risultante dalla somma dei contributi proporzionale e integrativo;
- il comando saturo  $u_{\text{sat}}$ , ottenuto limitando  $u_{\text{unsat}}$  all'intervallo ammesso  $[U_{\min}, U_{\max}]$ .

In questo modo, il segnale effettivamente applicato all'attuatore è sempre compatibile con i limiti del sistema, evitando comportamenti non fisici o non riproducibili.

## Integrator windup

Nel controllore PI, il termine integrativo accumula l'errore nel tempo con l'obiettivo di annullare l'errore a regime. Tuttavia, quando l'attuatore è saturo, il comando applicato non segue più la dinamica prevista dal controllore.

In tale condizione, l'errore può persistere e l'integratore continua ad accumularsi senza produrre un effetto reale sull'uscita del sistema.

Questo accumulo eccessivo dell'azione integrativa, noto come *integrator windup*, può causare tempi di assestamento significativamente più lunghi al rientro dalla saturazione.

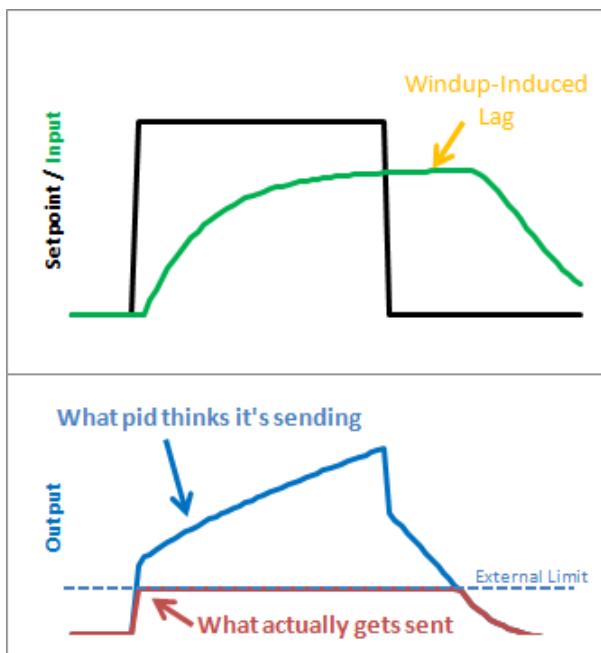


Figura 10.4: Integrator windup

**Soluzione**

Per evitare l'accumulo incontrollato dell'integratore nell'algoritmo implementato viene adottata una tecnica di *anti-windup* di tipo *back-calculation*. In tale approccio, l'aggiornamento del termine integrativo include una retroazione basata sulla differenza tra il comando teorico calcolato dal controllore e quello effettivamente applicato all'attuatore:

$$I(k+1) = I(k) + T_s \left( k_i e(k) + (u_{\text{sat}}(k) - u_{\text{unsat}}(k)) \right).$$

Il termine correttivo  $(u_{\text{sat}} - u_{\text{unsat}})$  realizza un meccanismo di *tracking* dell'uscita del controllore rispetto al segnale realmente applicato: esso risulta nullo quando il comando non è satura, mentre diventa significativo in condizioni di saturazione, contrastando la crescita dell'integratore e forzandone l'evoluzione coerente con i limiti dell'attuatore.

**Bump**

Il *bump* del comando di controllo è una variazione improvvisa e non desiderata dell'uscita del controllore, che si manifesta tipicamente in assenza di variazioni del riferimento o della dinamica dell'impianto. Tale fenomeno è riconducibile a una incoerenza tra lo stato interno del controllore, in particolare del termine integrativo, e il punto di funzionamento effettivo del sistema. In condizioni di regime stazionario, quando l'errore è nullo, il contributo proporzionale e quello derivativo risultano nulli e l'uscita del controllore è interamente determinata dal termine integrativo. Se il valore accumulato dall'integratore non corrisponde al comando necessario a mantenere il sistema in equilibrio, l'attivazione del controllo automatico o l'uscita da una condizione di saturazione può generare una discontinuità del comando applicato all'attuatore. Il problema del *bump* è particolarmente rilevante durante il passaggio dalla modalità degradata a quella Normal o al termine di una saturazione, poiché in questi stati il sistema può produrre sollecitazioni indesiderate sull'attuatore e transitori non controllati.

**Soluzione**

Nel sistema in esame, il problema del *bump* viene mitigato adottando la stessa strategia di *anti-windup* basata su *back-calculation* utilizzata per la gestione della saturazione. Infatti, il termine di *tracking* forza lo stato interno dell'integratore a seguire il valore del comando effettivamente applicato all'attuatore, mantenendo coerenza tra lo stato del controllore e il punto di funzionamento reale del sistema. Pertanto al rientro da una condizione di saturazione o al rientro dalla modalità degradata, il termine integrativo non introduce discontinuità improvvise nel comando.

# Implementazione software

In questo capitolo viene presentata l'implementazione software del sistema di controllo dei motori del rover, con particolare attenzione alle scelte progettuali adottate per l'interfacciamento con i driver di potenza e per la gestione dei segnali di comando. Vengono descritte le modalità operative dei driver Sabertooth e le procedure di calibrazione dei motori, finalizzate a garantire un comportamento uniforme e prevedibile del sistema di attuazione. Successivamente, il capitolo illustra la struttura dell'algoritmo di controllo, analizzandone le principali fasi operative, dalla generazione dei riferimenti alla produzione del comando PWM applicato agli attuatori. Infine, viene descritta la logica di gestione della sterzata, che consente di tradurre i comandi dell'utente nel comportamento cinematico desiderato del veicolo.

## 11.1 Scelta della modalità di pilotaggio dei driver Sabertooth

I driver di potenza Sabertooth 2x12 offrono diverse modalità operative per il comando dei motori in corrente continua, selezionabili tramite configurazione dei DIP switch a bordo scheda. In particolare, il manuale del costruttore prevede le seguenti modalità principali: *Analog Input*, *R/C Input*, *Simplified Serial* e *Packetized Serial*. Nel contesto del presente progetto, la scelta è ricaduta sulla modalità **Analog Input**, che è esplicitamente indicata dal costruttore come adatta alla realizzazione di *analog feedback loops* e all'uso con segnali PWM filtrati provenienti da microcontrollori, a condizione di adottare una frequenza PWM superiore a 1 kHz.

### Nota

L'adozione delle modalità seriali del Sabertooth (Simplified Serial o Packetized Serial) avrebbe comportato una latenza associata alla comunicazione UART, rendendo meno agevole la modellazione complessiva del sistema di controllo. La modalità analogica consente invece una rappresentazione continua del comando ed è intrinsecamente meno sensibile a disturbi di natura digitale, risultando pertanto la più adatta agli obiettivi del presente progetto.

Altre impostazioni tenute in considerazione relative alla Saberthoot sono state impostate mediante l'ausilio dei DIP switch posti sulla scheda del driver. In particolare:

- **DIP switch 3 (Lithium Cutoff)**: impostato in posizione *DOWN* per abilitare il cutoff automatico per batterie al litio. Poiché il sistema è alimentato da una batteria Li-ion a 3 celle (tensione nominale 12.6 V), tale funzione protegge interrompendo l'erogazione di potenza al raggiungimento della soglia minima di sicurezza.
- **DIP switch 4 (Independent Mode)**: impostato in posizione *DOWN* per selezionare la modalità *Independent*, nella quale ciascun ingresso analogico comanda direttamente un motore, lasciando al software di controllo la completa gestione della cinematica del rover.

- **DIP switch 6 (4x Sensitivity):** impostato in posizione *DOWN* per abilitare la modalità a sensibilità aumentata, che restringe l'intervallo utile del comando a  $1.875 \text{ V} \div 3.125 \text{ V}$  e migliora la risoluzione del controllo attorno al punto di arresto.

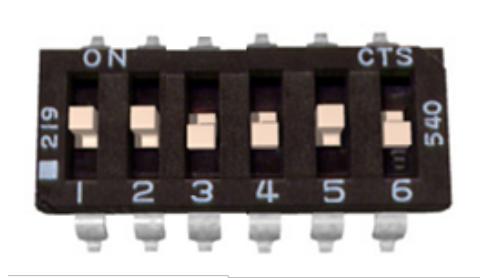


Figura 11.1: Configurazione dei DIP switch del driver Saberthoot

## 11.2 Calibrazione dei motori per il controllo in anello aperto

La calibrazione dei motori costituisce una fase preliminare fondamentale per garantire un comportamento coerente e prevedibile del sistema di attuazione del rover, indipendentemente dalla strategia di controllo adottata. Nella configurazione analogica ciascun canale di controllo interpreta un segnale di tensione compreso nell'intervallo da 1.875 V a 3.125 V, con un valore centrale pari a 2.5 V corrispondente alla condizione di arresto. Scostamenti positivi o negativi rispetto a tale valore determinano la rotazione del motore in senso orario o antiorario, con velocità proporzionale all'entità dello scostamento.

La relazione tra tensione applicata e velocità angolare risultante dipende dalle caratteristiche elettromeccaniche complessive del sistema motore–driver. Sebbene i motori installati sul rover siano nominalmente identici (stesso modello e costruttore), nella pratica essi presentano inevitabili variazioni parametriche dovute alle tolleranze di produzione. Per compensare tali differenze, è stata introdotta una procedura di calibrazione individuale per ciascun motore. La calibrazione consente di caratterizzare il comportamento di ogni attuatore e di introdurre opportuni fattori di scala sui comandi generati dal sistema di controllo, in modo da uniformare la risposta dei motori a fronte di comandi equivalenti. L'importanza di tale procedura risulta particolarmente evidente nel caso di comando in anello aperto, in cui l'assenza di retroazione impedisce qualsiasi correzione automatica delle discrepanze tra la velocità comandata e quella effettivamente ottenuta. In questa configurazione, la calibrazione rappresenta l'unico strumento disponibile per ridurre le asimmetrie di comportamento tra i motori e garantire un moto del veicolo il più possibile coerente con il comando impartito.

### 11.2.1 Calcolo della frequenza PWM

Il microcontrollore STM32 utilizzato nel progetto è configurato con un clock interno spinto a 170 MHz. Sebbene siano consigliate frequenze superiori a 1000 Hz per garantire un buon funzionamento del motore, come indicato nel manuale Sabertooth 2x12, è stata preferita una frequenza di 20 kHz rispetto a frequenze più basse, per garantire un controllo preciso senza produrre rumori udibili e ridurre il ripple nel segnale di potenza.

Per ottenere una frequenza di 20 kHz per il segnale PWM, i parametri del **prescaler** e del **ARR** (Auto-Reload Register) sono stati impostati come segue:

- Il **prescaler** è stato impostato a 1.
- L' **ARR** (Auto-Reload Register) è stato impostato a 4249.

La formula per calcolare la frequenza di uscita del timer è la seguente:

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

Dove:

- $f_{\text{clk}}$  = 170 MHz è la frequenza del clock del microcontrollore.
- Il **prescaler** è impostato a 1.
- Il **ARR** è impostato a 4249.

Sostituendo i valori:

$$f_{\text{PWM}} = \frac{170 \times 10^6}{(1 + 1) \times (4249 + 1)} = \frac{170 \times 10^6}{2 \times 4250} = 20 \text{ kHz}$$

La calibrazione è stata condotta sperimentalmente per ciascuno dei quattro motori del rover, seguendo una procedura standardizzata articolata in due fasi principali.

### 11.2.2 Fase 1: Determinazione del punto di zero

La determinazione del punto di zero è stata effettuata identificando il valore di PWM che, una volta convertito in tensione analogica tramite un filtro passa-basso, produce esattamente 2.5 V ai terminali di ingresso del driver. Per ottenere tale valore, abbiamo impostato inizialmente il valore del PWM come:

$$\text{PWM\_stop\_value} = \frac{2.5}{3.3} \times 4250 \approx 3220$$

Successivamente, il valore è stato regolato empiricamente finché la ruota non appariva immobile. I valori identificati per ciascun motore sono riportati nel file di intestazione `hw_calibration.h` con la nomenclatura `PWM_STOP_<motore>`, come segue:

```

1  /* ----- PWM stop ----- */
2  #define PWM_STOP_FA          (3230u)
3  #define PWM_STOP_FB          (3230u)
4  #define PWM_STOP_BA          (3215u)
5  #define PWM_STOP_BB          (3215u)
```

È importante osservare che, nonostante i motori siano del medesimo modello, i valori di PWM necessari per ottenere il punto di zero non sono identici.

### 11.2.3 Fase 2: Determinazione dei fattori di scala

Una volta stabilito il punto di zero, la seconda fase della calibrazione consiste nell'identificazione dei fattori di scala che relazionano il duty cycle del comando PWM alla velocità angolare del motore.

Per motivi di sicurezza e per garantire omogeneità nel movimento di tutti i motori la velocità massima ammessa è stata limitata via software a 150 RPM.

Per ciascun motore sono stati determinati sperimentalmente due fattori di scala distinti:

- **Fattore di scala in avanti (PWM\_SCALE\_FORWARD)**: quantità di PWM necessaria per incrementare la tensione dal punto di zero fino al valore che produce una rotazione in senso orario alla velocità di 150 RPM.
- **Fattore di scala all'indietro (PWM\_SCALE\_BACKWARD)**: quantità di PWM necessaria per decrementare la tensione dal punto di zero fino al valore che produce una rotazione in senso antiorario alla velocità di 150 RPM.

La determinazione sperimentale per i parametri di scala è stata condotta attivando ciascun motore singolarmente in modalità open loop e acquisendo tramite encoder la velocità effettiva raggiunta a regime.

```

1  /* ----- PWM scale forward ----- */
2  #define PWM_SCALE_FORWARD_FA      (690u)
3  #define PWM_SCALE_FORWARD_FB      (710u)
4  #define PWM_SCALE_FORWARD_BA      (690u)
5  #define PWM_SCALE_FORWARD_BB      (690u)
6
7  /* ----- PWM scale backward ----- */
8  #define PWM_SCALE_BACKWARD_FA    (640u)
9  #define PWM_SCALE_BACKWARD_FB    (630u)
10 #define PWM_SCALE_BACKWARD_BA    (630u)
11 #define PWM_SCALE_BACKWARD_BB    (620u)

```

I fattori di scala così ottenuti sono stati memorizzati nel file `hw_calibration.h`.

## 11.3 Algoritmo di controllo

In questa sezione si commenta passo passo la procedura per il controllo in retroazione dei motori. Come già ricordato nel paragrafo 10.2.8 l'algoritmo di controllo viene eseguito ad ogni tempo di campionamento (ogni 5ms) secondo la seguente sequenza operativa:

1. Acquisizione dei segnali di uscita dell'impianto mediante conversione analogico-digitale;
2. Calcolo del segnale di comando sulla base dell'algoritmo di controllo discretizzato;
3. Invio del comando all'attuatore mediante conversione digitale-analogica.

### 11.3.1 Acquisizione del segnale

Dunque, anzitutto per ciascun encoder viene invocata la funzione `encoder_readRPM`.

```

1 if (encoder_readRPM(&encoder_FA_pid) != ENCODER_OK) {
2     Error_Handler();
3 if (encoder_readRPM(&encoder_FB_pid) != ENCODER_OK) {
4     Error_Handler();
5 if (encoder_readRPM(&encoder_BA_pid) != ENCODER_OK) {
6     Error_Handler();
7 if (encoder_readRPM(&encoder_BB_pid) != ENCODER_OK) {
8     Error_Handler();

```

**encoder\_readRPM** La funzione legge il contatore del timer e calcola la differenza rispetto alla lettura precedente:

```

1 uint32_t cnt = __HAL_TIM_GET_COUNTER(enc->htim);
2
3 if(enc->prev_count == cnt){
4     diff = 0;
5     enc->velocity = 0;
6 }
```

Se il contatore non è cambiato, allora nell'intervallo di campionamento non ci sono stati impulsi encoder: la velocità viene posta a zero. Il calcolo della differenza dipende dal verso di conteggio del timer:

```

1 else if (__HAL_TIM_IS_TIM_COUNTING_DOWN(enc->htim)) {
2
3     if (cnt < enc->prev_count) diff = enc->prev_count - cnt;
4     else diff = (enc->arr - cnt) + enc->prev_count;
5
6     enc->velocity = (60.0f * diff) /
7                         (enc->calib.pulses_per_rev * ...
8                          enc->reading_period * enc->calib.gear_ratio);
9 }
```

Se il timer sta contando in down-counting, la differenza `diff` viene calcolata gestendo anche l'overflow (wrap sul registro autoreload `arr`). La conversione in RPM avviene moltiplicando per 60 e dividendo per `pulses_per_rev` (impulsi per giro), `reading_period` (periodo in secondi), `gear_ratio` (rapporto di riduzione).

Nel caso di conteggio in up-counting, il segno viene invertito:

```

1 else {
2
3     if (cnt > enc->prev_count) diff = cnt - enc->prev_count;
4     else diff = (enc->arr - enc->prev_count) + cnt;
5
6     enc->velocity = -(60.0 * diff) /
7                         (enc->calib.pulses_per_rev * ...
8                          enc->reading_period * enc->calib.gear_ratio);
9 }
```

Qui `enc->velocity` è resa negativa, quindi il segno della velocità è coerente con il verso di rotazione. Infine viene gestita una correzione legata alla modalità encoder:

```

1 if ((enc->htim->Instance->SMCR & 0x3) == 0x3)
2     enc->velocity /= 2;
3
4 enc->prev_count = __HAL_TIM_GET_COUNTER(enc->htim);
```

La condizione sul registro SMCR verifica una particolare configurazione hardware; in tal caso la velocità viene divisa per 2. Alla fine, `prev_count` viene aggiornato con il contatore corrente per la prossima iterazione.

### 11.3.2 Ramping del riferimento

I riferimenti di velocità generati dal livello di supervisione non vengono applicati direttamente ai regolatori PID, ma sono preventivamente filtrati mediante un meccanismo

di *ramping*, che limita la pendenza massima delle variazioni del riferimento tra due campioni consecutivi. In particolare:

**Nel caso di frenata di emergenza**, la priorità assoluta è ridurre il più rapidamente possibile la velocità del rover, minimizzando lo spazio di arresto. In questa condizione, qualsiasi limitazione artificiale sulla pendenza del riferimento risulterebbe controproducente. Per tale motivo, quando il sistema entra in modalità **EMERGENCY** e tutti i riferimenti di velocità richiesti sono nulli, i riferimenti rampati vengono forzati istantaneamente al valore di target:

```

1 if (braking_mode == EMERGENCY && (rif_FA == 0 && rif_FB == 0 && ...
2   rif_BA == 0 && rif_BB == 0)) {
3
4   rif_FA_r = rif_FA;
5   rif_FB_r = rif_FB;
6   rif_BA_r = rif_BA;
7   rif_BB_r = rif_BB;
8
9 }
```

**Nel caso di frenata smooth**, l'obiettivo non è arrestare il veicolo nel minor tempo possibile, ma ottenere una decelerazione controllata: in questa modalità, il ramping viene mantenuto, ma con una pendenza ridotta, ottenuta moltiplicando il passo di rampa per un coefficiente di frenata:

```

1 else if (braking_mode == NORMAL &&
2   (rif_FA == 0 && rif_FB == 0 && rif_BA == 0 && rif_BB == 0)) {
3   rif_FA_r = ramp(rif_FA_r, rif_FA, ramp_step * NORMAL_BRK_COEFF);
4   rif_FB_r = ramp(rif_FB_r, rif_FB, ramp_step * NORMAL_BRK_COEFF);
5   rif_BA_r = ramp(rif_BA_r, rif_BA, ramp_step * NORMAL_BRK_COEFF);
6   rif_BB_r = ramp(rif_BB_r, rif_BB, ramp_step * NORMAL_BRK_COEFF);
7 }
```

In tutte le altre condizioni operative, viene applicata la rampa standard:

```

1 else {
2   rif_FA_r = ramp(rif_FA_r, rif_FA, ramp_step);
3   rif_FB_r = ramp(rif_FB_r, rif_FB, ramp_step);
4   rif_BA_r = ramp(rif_BA_r, rif_BA, ramp_step);
5   rif_BB_r = ramp(rif_BB_r, rif_BB, ramp_step);
6 }
```

**ramp** Il comportamento del ramping è implementato dalla seguente funzione:

```

1 real32_T ramp(real32_T current, real32_T target, real32_T step)
2 {
3   if (current < target - step){
4     return current + step;
5   }
6   else if (current > target + step){
7     return current - step;
8   }
9   else{
10     return target;
11   }
12 }
```

La funzione garantisce che la variazione del riferimento tra due campioni consecutivi non superi il valore `step`. In particolare:

- se il valore corrente è significativamente inferiore al target, il riferimento aumenta di una quantità fissa `step`;
- se è significativamente superiore, diminuisce della stessa quantità;
- se la differenza è inferiore a `step`, il riferimento viene allineato direttamente al target, evitando oscillazioni residue.

### 11.3.3 Calcolo del comando PID

Una volta ottenuto il riferimento rampato `rif_X_r` e la misura di velocità fornita dall'encoder `encoder_X_pid.velocity`, il comando di controllo viene calcolato mediante il regolatore PID digitale associato al motore.

L'operazione è eseguita tramite la chiamata:

```
1 PID_compute(&pid_FA, rif_FA_r, encoder_FA_pid.velocity, &control_FA;
2 PID_compute(&pid_FB, rif_FB_r, encoder_FB_pid.velocity, &control_FB;
3 PID_compute(&pid_BA, rif_BA_r, encoder_BA_pid.velocity, &control_BA;
4 PID_compute(&pid_BB, rif_BB_r, encoder_BB_pid.velocity, &control_BB;
```

**PID\_compute** Il compito della funzione `PID_compute` è determinare, ad ogni campione, il valore del comando di controllo a partire dall'errore tra riferimento e uscita misurata, tenendo conto dei vincoli fisici dell'attuatore.

Il primo passo consiste nel calcolo dell'errore istantaneo:

```
1 float error = setpoint - feedback;
2 float P = pid->kp * error;
```

L'errore rappresenta la differenza tra la velocità desiderata e quella effettivamente misurata. Il termine proporzionale fornisce una correzione immediata, proporzionale all'ampiezza dell'errore, ed è responsabile della prontezza della risposta del sistema.

Il contributo proporzionale viene sommato al termine integrativo accumulato:

```
1 float u_unsat = P + pid->I_term;
2 float u_sat = u_unsat;
```

Il valore `u_unsat` rappresenta il comando teorico calcolato dal PID in assenza di vincoli. Come esposto nel paragrafo 10.2.9, l'attuatore reale è soggetto a limiti fisici e di progetto, per cui il comando deve essere limitato all'intervallo ammesso:

```
1 if (u_unsat > U_MAX) u_sat = U_MAX;
2 else if (u_unsat < U_MIN) u_sat = U_MIN;
```

Il valore `u_sat` è quindi il comando effettivamente applicabile all'attuatore, compatibile con i limiti del sistema. Come esposto nel paragrafo 10.2.9, per evitare che il termine integrativo continui a crescere quando il comando è saturo (fenomeno di *integrator windup*), l'integratore viene aggiornato introducendo una correzione basata sulla differenza tra comando saturato e non saturato:

```
1 pid->I_term += pid->period *
2 (pid->ki * error + (u_sat - u_unsat));
```

Il termine  $(u_{sat} - u_{unsat})$  agisce come retroazione sull'integratore: quando il comando è saturo, la differenza è non nulla e limita la crescita dell'integrale. In questo modo:

- si evita l'accumulo inutile di errore;
- si riducono sovraelongazioni al rientro dalla saturazione;

Il contributo derivativo è calcolato sulla misura di velocità, anziché sull'errore, per evitare amplificazione del rumore in presenza di variazioni brusche del riferimento:

```
1 float d_raw = (feedback - pid->prev_feedback) / pid->period;
```

Poiché la derivata discreta è particolarmente sensibile al rumore di misura, essa viene filtrata mediante un filtro passa-basso del primo ordine:

```
1 float alpha = pid->tau / (pid->tau + pid->period);
2 pid->d_filt = alpha * pid->d_filt + (1.0f - alpha) * d_raw;
3
4 float D = pid->kd * pid->d_filt;
```

Il parametro `tau` rappresenta la costante di tempo del filtro: valori maggiori di `tau` producono un filtraggio più lento e una maggiore attenuazione del rumore, a scapito della reattività.

Il comando finale viene ottenuto sottraendo il contributo derivativo dal comando saturato:

```
1 *u_out = u_sat - D;
2 pid->prev_feedback = feedback;
```

La sottrazione del termine derivativo consente di contrastare variazioni rapide della velocità misurata, introducendo un effetto di smorzamento che migliora la stabilità del sistema.

#### Nota

Come discusso nel paragrafo 10.2.1, sebbene l'algoritmo di controllo sia stato implementato nella sua forma più generale di PID, includendo esplicitamente il calcolo e il filtraggio del termine derivativo, la sintesi del controllore per il sistema in esame è stata condotta scegliendo un controllore di tipo PI.

Di conseguenza, per tutti i controllori utilizzati nel sistema il guadagno derivativo è posto pari a zero ( $k_d = 0$ ). In tali condizioni il contributo derivativo risulta nullo e la relativa parte di codice, pur rimanendo presente per motivi di generalità e riusabilità dell'algoritmo, non influisce in alcun modo sul comando di controllo effettivamente applicato al motore.

### 11.3.4 Applicazione al motore: saturazione duty e calcolo PWM

L'uscita del regolatore PID è un valore continuo e firmato, che rappresenta il comando di controllo in unità normalizzate. Prima di poter essere applicato all'attuatore, tale comando deve essere convertito in una forma compatibile con l'interfaccia del driver motore, basata su un segnale PWM.

Nel task di controllo, questa conversione avviene mediante la seguente operazione:

```
1 out_FA = abs(round(control_FA * MOTOR_MAX_DUTY / U_MAX));
2 out_FB = abs(round(control_FB * MOTOR_MAX_DUTY / U_MAX));
3 out_BA = abs(round(control_BA * MOTOR_MAX_DUTY / U_MAX));
4 out_BB = abs(round(control_BB * MOTOR_MAX_DUTY / U_MAX));
```

Il valore `control_X` è un comando prodotto dal PID:

- il *modulo* rappresenta l'intensità del comando;
- il *segno* codifica la direzione di rotazione desiderata.

La normalizzazione rispetto a `U_MAX` consente di mappare l'intervallo di uscita del PID sull'intervallo ammesso per il duty-cycle, definito da `MOTOR_MAX_DUTY`. In questo modo, una saturazione del PID corrisponde direttamente a un duty massimo applicabile al motore. L'operazione di `round` discretizza il comando continuo al livello richiesto dall'hardware PWM, mentre la funzione `abs` garantisce che il duty-cycle sia sempre non negativo, come richiesto dal driver. La direzione di rotazione del motore viene determinata separatamente, a partire dal segno del comando:

```
1 (control_FA > 0) ? CLOCKWISE : COUNTERCLOCKWISE
2 (control_FB > 0) ? CLOCKWISE : COUNTERCLOCKWISE
3 (control_BA > 0) ? CLOCKWISE : COUNTERCLOCKWISE
4 (control_BB > 0) ? CLOCKWISE : COUNTERCLOCKWISE
```

Una volta determinati il duty-cycle e la direzione di rotazione, il task di controllo invoca il driver motore:

```
1 motor_set(&motor_FA, out_FA,
2           (control_FA > 0) ? CLOCKWISE : COUNTERCLOCKWISE);
3
4 motor_set(&motor_FB, out_FB,
5           (control_FB > 0) ? CLOCKWISE : COUNTERCLOCKWISE);
6
7 motor_set(&motor_BA, out_BA,
8           (control_BA > 0) ? CLOCKWISE : COUNTERCLOCKWISE);
9
10 motor_set(&motor_BB, out_BB,
11            (control_BB > 0) ? CLOCKWISE : COUNTERCLOCKWISE);
```

**motor\_set** La funzione `motor_set` realizza la traduzione finale del comando astratto in un segnale PWM fisico, applicato al timer hardware associato al motore. Questa operazione avviene in tre fasi distinte.

**(i) Clipping del duty-cycle** Il primo passo consiste nella saturazione esplicita del duty:

```
1 static inline uint8_t clip_duty(uint8_t duty){
2     if (duty > MOTOR_MAX_DUTY){
3         return MOTOR_MAX_DUTY;
4     }
5     return duty;
6 }
```

Questo controllo introduce una protezione aggiuntiva, indipendente dal PID, che garantisce che il valore di duty applicato non superi mai i limiti ammissibili dall'hardware. In questo modo si rende il driver robusto anche in presenza di errori a monte.

**(ii) Conversione del duty in valore PWM assoluto** Il duty normalizzato viene successivamente convertito nel valore PWM assoluto da scrivere nel registro di compare del timer:

```

1 static uint16_t compute_pwm(const Motor_t* m,
2                             uint8_t duty,
3                             Motor_Direction_t dir){
4     uint16_t pwm;
5
6     if (duty == MOTOR_MIN_DUTY){
7         pwm = m->calib.pwm_stop;
8     }
9     else if (dir == CLOCKWISE){
10        pwm = m->calib.pwm_stop +
11            (uint16_t)roundf((float)duty *
12                m->calib.pwm_scale_forward / MOTOR_MAX_DUTY);
13    }
14    else if (dir == COUNTERCLOCKWISE){
15        pwm = m->calib.pwm_stop -
16            (uint16_t)roundf((float)duty *
17                m->calib.pwm_scale_backward / MOTOR_MAX_DUTY);
18    }
19    else{
20        pwm = m->calib.pwm_stop;
21    }
22
23    return pwm;
24 }
```

Il parametro `pwm_stop` rappresenta il valore PWM che corrisponde alla condizione di arresto del motore. I coefficienti `pwm_scale_forward` e `pwm_scale_backward` sono i fattori di scala necessari per incrementare la tensione dal punto di zero fino al valore che produce la massima velocità nel senso orario/antiorario. Tali valori sono ottenuti in fase di calibrazione e sono specifici per ciascun attuatore (vedi paragrafo 11.2).

**(iii) Applicazione del PWM al timer hardware** Infine, il valore PWM calcolato viene applicato direttamente al canale del timer associato al motore:

```

1 static inline void apply_pwm(const Motor_t* m, uint16_t value){
2     __HAL_TIM_SET_COMPARE(m->htim, m->channel, value);
3 }
```

La scrittura nel registro di compare modifica istantaneamente il duty-cycle del segnale PWM generato dal timer, traducendo il comando logico in una tensione media applicata all'ingresso analogico del driver di potenza.

## 11.4 Gestione della sterzata

Per tradurre i comandi di guida forniti dall'utente, tramite joystick in riferimenti di velocità angolare espressi in RPM per ciascuno dei quattro motori del rover abbiamo una funzione di utilità posto a monte dell'algoritmo di controllo, essa costituisce un modulo di utilità posto a monte dell'algoritmo di controllo dei motori. Essa non svolge alcuna azione di controllo dinamico, ma definisce esclusivamente il comportamento cinematico desiderato del veicolo a partire dall'interazione dell'utente.

La funzione inizia selezionando la velocità massima ammessa, che rappresenta il limite superiore per i riferimenti generati:

```
1 MAX_SPEED = single(global_state.max_vel);
```

Questo parametro consente di adattare dinamicamente il comportamento del rover a diversi profili di guida senza modificare la logica dell'algoritmo.

Gli input del joystick sono acquisiti come valori interi centrati attorno a una posizione di riposo. Tali valori vengono prima traslati rispetto al centro e poi normalizzati:

```
1 diff_x = single(global_state.stateB2.controller_x) - single(CENTER);
2 diff_y = single(global_state.stateB2.controller_y) - single(CENTER);
3
4 steering = single(diff_x) / CENTER;
5 throttle = single(diff_y) / CENTER;
```

In questo modo, i comandi di sterzata e accelerazione vengono espressi come grandezze adimensionali comprese tra  $-1$  e  $+1$ , rendendo l'algoritmo indipendente dal range numerico specifico del dispositivo di input.

A partire dagli input normalizzati, vengono calcolate due componenti fondamentali:

```
1 forward = throttle * MAX_SPEED;
2 turn = steering * MAX_SPEED;
```

La variabile `forward` rappresenta la velocità longitudinale desiderata, mentre `turn` rappresenta il contributo di rotazione. Questa scomposizione consente di trattare separatamente avanzamento e sterzata

L'algoritmo distingue esplicitamente il caso di rotazione sul posto dal caso di curva in movimento:

```
1 if abs(throttle) < PURE_TURN_EPS
2     forward = single(0);
3 else
4     max_turn = abs(forward) * TURN_RATIO;
5     if abs(turn) > max_turn
6         turn = sign(turn) * max_turn;
7     end
8 end
```

Quando il comando longitudinale è nullo, il rover ruota su sé stesso. In caso contrario, l'ampiezza della sterzata viene limitata in funzione della velocità di avanzamento, evitando variazioni di traiettoria troppo brusche.

Le velocità del lato sinistro e destro del veicolo sono ricavate tramite una combinazione differenziale dei comandi.

```
1 left = forward + turn;
2 right = forward - turn;
```

Questa operazione realizza la cinematica tipica di un veicolo a trazione differenziale.

Per garantire il rispetto dei limiti fisici dei motori, viene applicata una normalizzazione proporzionale:

```
1 maxabs = max(abs(left), abs(right));
2 if maxabs > MAX_SPEED
3     scale_factor = MAX_SPEED / maxabs;
4     left = left * scale_factor;
5     right = right * scale_factor;
6 end
```

Entrambi i riferimenti vengono scalati dello stesso fattore, preservando il rapporto tra i due lati e quindi il raggio di curvatura desiderato.

Infine, i riferimenti calcolati vengono assegnati ai quattro motori:

```
1 decision.rif_FA = left;
2 decision.rif_BA = left;
3 decision.rif_FB = right;
4 decision.rif_BB = right;
```

I motori sullo stesso lato ricevono lo stesso riferimento di velocità, garantendo un comportamento coerente del sistema di trazione.

# Comando e gestione remota

Il sistema di comando e gestione remota si avvale di tre diversi microcontrollori ESP32, due dei quali installati direttamente a bordo del veicolo, mentre il terzo contenuto all'interno del controller realizzato “ad-hoc” per il rover. Le diverse entità dialogano tra di loro avvalendosi, come è possibile osservare in figura, di diversi protocolli di comunicazione. Nei paragrafi seguenti analizzeremo in dettaglio ognuno di essi.

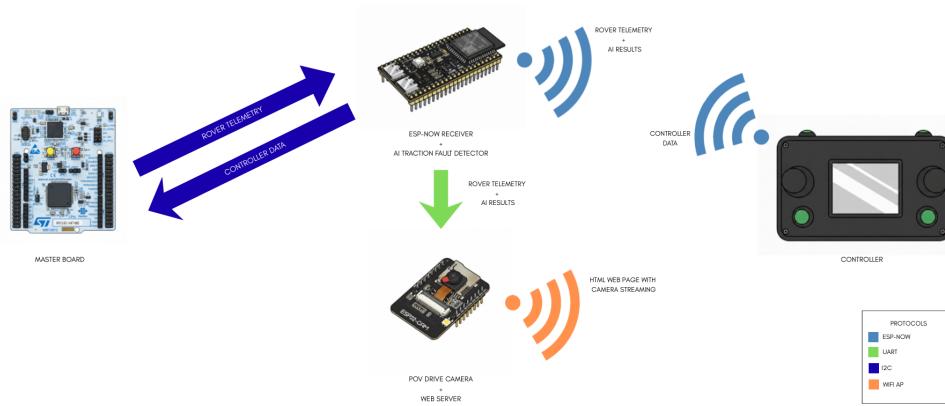


Figura 12.1: Architettura per il controllo in remoto

## 12.1 Il controller

Lo sviluppo del controller per il rover è stato concepito con l'obiettivo primario di realizzare rapidamente un prototipo funzionale. Per tale ragione, la scelta è ricaduta sul framework Arduino, dimostratosi ideale per accelerare la creazione del dispositivo di controllo.

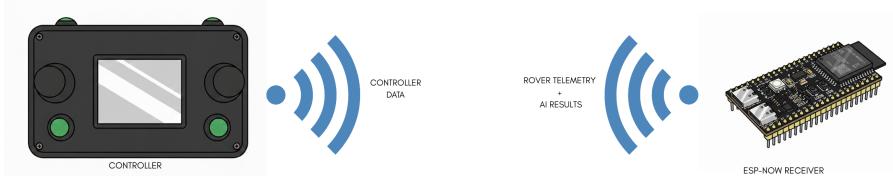
### 12.1.1 Componenti del controller

Nella sua configurazione attuale, il controller è equipaggiato con due joystick analogici, destinati al controllo direzionale, e quattro pulsanti digitali, dedicati alle funzioni ausiliarie. L'interfaccia utente è mostrata su una dev board modello ESP32-2432S028R, che integra un display touchscreen da 2.8 pollici. Questo schermo non solo fornisce un feedback visivo all'operatore, ma costituisce anche il canale principale di comunicazione con la scheda YD ESP32-S3 DevKit installata a bordo del rover.

### 12.1.2 Protocollo di comunicazione

La comunicazione tra controller e rover è gestita attraverso il protocollo ESP-NOW. Questa è una soluzione proprietaria di Espressif, preferita rispetto a standard più diffusi come Wi-Fi o Bluetooth Low Energy (BLE) per la sua intrinseca semplicità e la superiore portata

che è in grado di offrire. Tecnicamente, ESP-NOW sfrutta lo stack fisico Wi-Fi della scheda, operando in modalità connectionless per l'invio dei pacchetti di dati, simulando di fatto una trasmissione radio generica. Il joystick instaura una comunicazione bidirezionale con il rover: invia i comandi generati da joystick e pulsanti e riceve le telemetrie che vengono poi visualizzate sul display. Nonostante la natura connectionless del protocollo, la ricevente a bordo del rover è dotata di un failsafe watchdog che si attiva se il dispositivo smette di inviare valori entro un tempo di timeout configurabile.



### 12.1.3 Gestione I/O aggiuntiva

Per ovviare al limitato numero di pin GPIO disponibili sul modulo ESP32 principale, il dispositivo integra una seconda dev-board, una YD-RP 2040, che funge da I/O expander e preprocessor. L'RP2040 è specificamente incaricato dell'elaborazione degli ingressi analogici. Gestisce la dead-zone centrale dei joystick, eliminando gli effetti di step indesiderati. Per assicurare una latenza minima, la comunicazione con l'ESP32 avviene tramite I2C ad alta velocità (400KHz).

### 12.1.4 Alimentazione e monitoraggio

Il sistema di alimentazione del controller si basa su una cella al litio 18650, affiancata da un circuito di ricarica dedicato e da un regolatore di tensione. Quest'ultimo fornisce una tensione stabile di 5V, essenziale per alimentare correttamente le due dev board. Per monitorare lo stato di carica della batteria, viene impiegato un partitore resistivo. Questo circuito è indispensabile per attenuare la tensione della batteria a un livello sicuro e compatibile con l'ADC dell'ESP32.

### 12.1.5 Interfaccia Grafica Utente (GUI)

La GUI di sistema è stata sviluppata utilizzando la libreria LVGL 9 e organizza le informazioni su tre schermate distinte:

- **Rover Status:** Mostra i dati relativi alla batteria del rover, la temperatura della scheda slave, lo stato della retromarcia e delle luci, la modalità di percezione degli ostacoli, lo stato della supervisione distribuita e la diagnosi AI del sistema.
- **Motor Status:** Visualizza il valore degli RPM per i quattro motori del sistema.
- **Sonar Status:** Indica la distanza percepita dai sensori a ultrasuoni anteriori.

In tutte le schermate, nella parte superiore, è presente una barra di stato che indica il livello di carica della batteria del telecomando. Il colore di questa barra varia in base alla modalità di guida configurata.

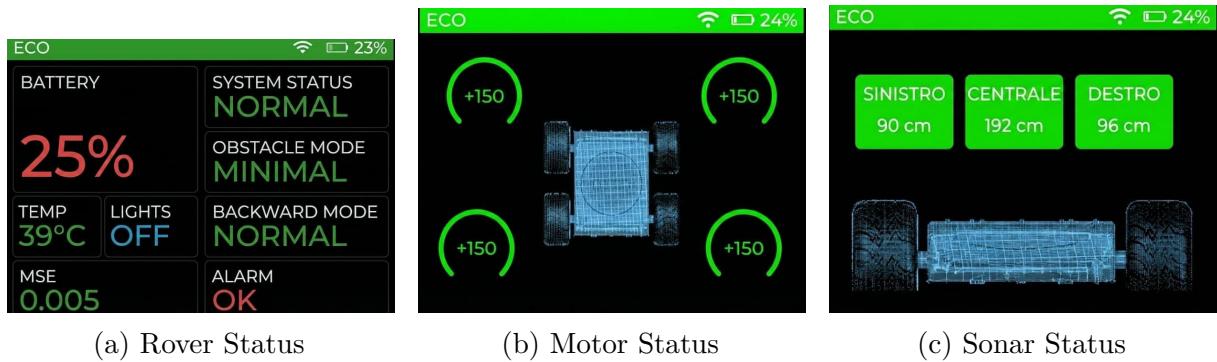


Figura 12.2: Schermate principali dell’interfaccia di monitoraggio del rover

Il firmware è strutturato su FreeRTOS per gestire la concorrenza operativa attraverso tre task dedicati: uno per l’interfaccia utente (GUI), uno per l’acquisizione dati via I2C dall’RP2040 e un terzo per la trasmissione delle telemetrie tramite ESP-NOW. L’impiego di strutture Queue garantisce l’eliminazione completa delle race condition tra i task. Il task di trasmissione ESP-NOW attende che la coda contenga i nuovi dati del controller letti via I2C. In assenza di dati entro un timeout prestabilito, il task rispedisce i comandi precedenti per prevenire l’attivazione del watchdog sulla ricevente, che altrimenti rileverebbe una perdita di connessione. Se la lettura I2C fallisce, il task invia comandi neutrali per garantire la sicurezza e prevenire situazioni pericolose. Viceversa, i pacchetti di telemetria ricevuti da ESP-NOW vengono inseriti in una coda dalla quale il task della UI preleva gli aggiornamenti più recenti da visualizzare. Il sistema di comando e gestione remota si avvale di tre diversi microcontrollori ESP32, due dei quali installati direttamente a bordo del veicolo, mentre il terzo contenuto all’interno del controller realizzato “ad-hoc” per il rover. Le diverse entità dialogano tra di loro avvalendosi, come è possibile osservare in figura, di diversi protocolli di comunicazione.

## 12.2 La ricevente

Lo sviluppo della ricevente ha avuto l’obiettivo di realizzare un sistema robusto, modulare e scalabile, capace di integrare molteplici sottosistemi in un’architettura orientata agli oggetti. La ricevente ha un ruolo cruciale nel sistema, essendo responsabile della gestione del telecontrollo e dell’acquisizione telemetrica.

### 12.2.1 Configurazione hardware

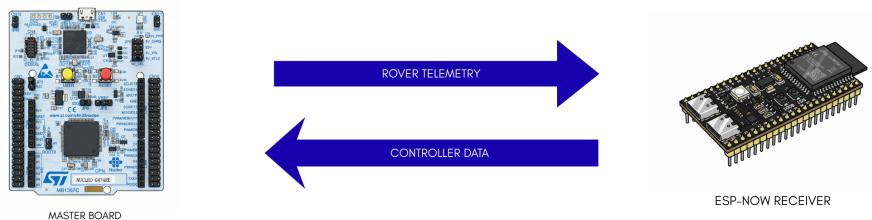
La ricevente è basata su una scheda YD ESP32-S3 DevKit, scelta per le sue elevate prestazioni di elaborazione e la capacità di supportare operazioni multi-core critiche per il sistema. La scheda integra due core di elaborazione indipendenti, sfruttati per parallelizzare i compiti più esigenti dal punto di vista computazionale.

### 12.2.2 Interfacce di comunicazione

La ricevente comunica con tre sottosistemi distinti attraverso altrettante interfacce dedicate:

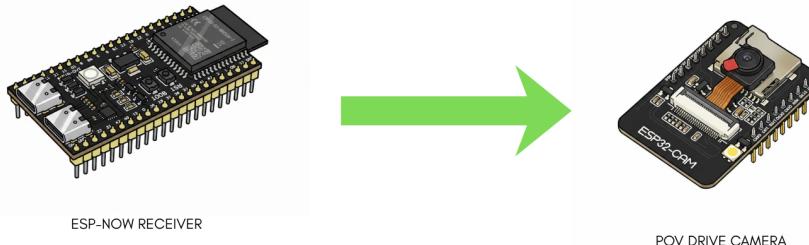
### I2C - Collegamento con la scheda ST Master

La comunicazione I2C rappresenta il canale principale di acquisizione dei dati sensoriali. Opera a velocità di 100 kHz e consente di inviare i comandi e di ricevere telemetrie dal sistema di controllo distribuito, il quale in condizione operative normali aggrega i dati provenienti dalle due schede di controllo principali. La ricevente è configurata come slave I2C, così che il supervisore del rover possa richiedere i comandi impartiti secondo sue necessità e secondo il suo ciclo di funzionamento.



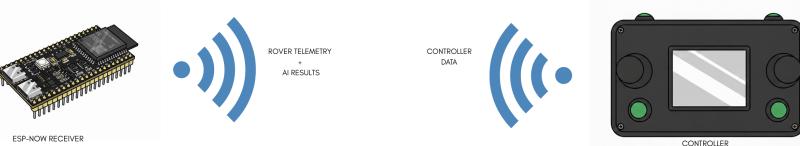
### UART - Telemetria verso ESP-CAM

Il collegamento UART, operante a 115200 baud, consente la trasmissione di telemetrie complete verso il modulo camera + web server. La ricevente invia pacchetti strutturati contenenti i dati del rover.



### ESP-NOW - Ricezione comandi dal controller ed invio delle telemetrie

La comunicazione wireless con il controller joystick avviene attraverso il protocollo proprietario ESP-NOW. Come nella controparte, questa soluzione è stata preferita per la sua semplicità, la bassa latenza e la superiore portata. La comunicazione avviene in modalità bidirezionale, consentendo al controller di inviare dati di controllo alla ricevente e di ricevere le telemetrie complete da mostrare sulla GUI menzionata in precedenza.



#### 12.2.3 Architettura firmware

Il firmware della ricevente è strutturato secondo principi di progettazione OOP, suddividendo le responsabilità in oggetti specializzati:

### ControllerManager - Gestione dello stato del controller

Questo oggetto mantiene lo stato dei comandi ricevuti dal joystick e implementa un meccanismo di failsafe discusso nel paragrafo precedente. Monitora il timestamp dell'ultimo pacchetto ricevuto e, qualora il controller rimanga silenzioso per più di un certo tempo limite (1500 ms), attiva automaticamente un timeout, impostando tutti i comandi sulla posizione neutrale. Questo garantisce che il rover non continui a eseguire comandi obsoleti in caso di perdita di connessione.

### TelemetryManager - Acquisizione e sincronizzazione telemetrica

Coordina la ricezione dei dati sensoriali via I2C dalla scheda NUCLEO master e fornisce accesso thread-safe ai dati telemetrici. Utilizza mutex critici e event group FreeRTOS per sincronizzare l'accesso ai dati tra task concorrenti, prevenendo race condition e corruzione dati.

### Communication Managers - Gestione multiprotocollo

Tre gestori di comunicazione specializzati controllano le tre interfacce hardware:

- **I2CComm:** Implementa uno slave I2C, rispondendo alle richieste della scheda master con gli ultimi dati del controller e ricevendo dati telemetrici.
- **UARTComm:** Serializza e trasmette i pacchetti telemetrici completi verso la stazione base.
- **WirelessComm:** Riceve i pacchetti di controllo ESP-NOW e li distribuisce al ControllerManager.

#### 12.2.4 Elaborazione concorrente

Il firmware impiega FreeRTOS per organizzare tre task principali distribuiti in:

##### Loop Principale (Core 1) – Watchdog e supervisione

Eseguito sul core 1 (utilizzato da Arduino per il setup e il loop), implementa la logica di watchdog per il controller. Con periodicità configurabile, verifica se il controller ha superato il timeout e, in caso affermativo, attiva il failsafe.

##### Task Telemetry Pipeline (Core 0) – Gestione delle telemetrie del rover

Dedicato al core 0, questo task elabora continuamente i dati telemetrici inviandoli alla ESP-CAMERA tramite protocollo UART ed al controller tramite ESP-NOW.

## 12.3 Visione e monitoraggio

Il sottosistema di visione del rover, basato su un modulo AI-THINKER ESP32-CAM, è stato concepito come un nodo per la guida remota e il monitoraggio dei parametri di funzionamento del rover.

### 12.3.1 Architettura

L'ESP32-CAM, grazie al suo firmware, opera come un server web per consentire la visualizzazione di una dashboard contenente i dati telemetrici e il flusso video. Per garantire l'operatività in scenari privi di infrastruttura di rete, il modulo configura il WiFi in modalità Access Point (AP). Il rover stabilisce la propria rete locale ("CRIC\_rover"), consentendo la connessione diretta di dispositivi esterni. La sincronizzazione dei dati telemetrici avviene comunicando con l'unità ricevente ESP-NOW tramite un'interfaccia UART a 115 200 baud. Per assicurare l'affidabilità in presenza di disturbi, è implementato un protocollo di comunicazione dotato di Framing di Sicurezza, marcando l'inizio e la fine di ogni pacchetto con dei "Magic Bytes" (0xCD, 0xAB), permettendo il riallineamento immediato del flusso dati in caso di desincronizzazione.

### 12.3.2 Gestione Concorrente con FreeRTOS

L'architettura software impiega il sistema operativo FreeRTOS per l'esecuzione parallela dei compiti critici:

- **Serial Task (Core 0):** Dedicato alla scansione continua del buffer UART per acquisire i pacchetti di telemetria.
- **Web/System Task:** Gestisce lo streaming video MJPEG e le richieste HTTP asincrone.

La comunicazione tra i task è gestita tramite una Queue a elemento singolo con modalità *Overwrite*. Questa strategia assicura che l'interfaccia web visualizzi sempre l'informazione più recente sullo stato del rover, scartando i dati obsoleti.

### 12.3.3 Dashboard per FPV

L'interfaccia utente è una Single Page Application (SPA) ottimizzata e integrata nella memoria flash (PROGMEM) per massimizzare la velocità. La dashboard è concepita per la guida remota (FPV) e presenta le informazioni in modo organizzato, circondando il flusso video alla stregua di un cockpit. Infine, la dashboard salva localmente nel browser tutte le telemetrie ricevute, consentendo di esportare in un file CSV quanto registrato.

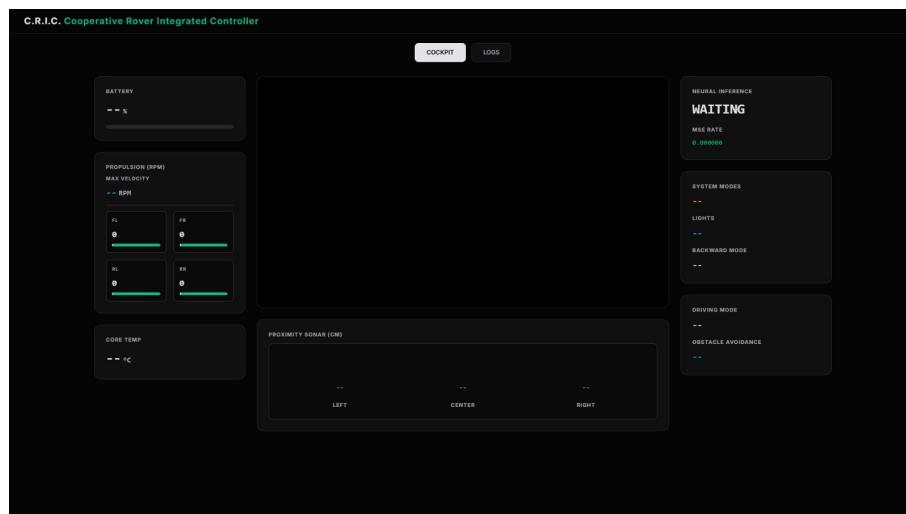


Figura 12.3: Dashboard Webserver

## **Parte V**

# **Dettagli implementativi e Test di sistema**

---

# Struttura del Firmware

---

Nel firmware del rover, l'architettura è organizzata su due board cooperanti, con una chiara separazione delle responsabilità, al fine di garantire robustezza, leggibilità e manutenibilità. Entrambe le board condividono lo stesso scheletro software, basato su **STM32 HAL** e **FreeRTOS**: l'inizializzazione delle periferiche è concentrata in `main.c`, mentre la definizione e la gestione dei task periodici avvengono in `app_freertos.c`. La logica applicativa di alto livello è implementata nei moduli `Board1.c` e `Board2.c`, generati automaticamente da Simulink secondo un approccio di *Model-Based Design*. I file `board1_functions.c` e `board2_functions.c` contengono invece le funzioni di interfaccia hardware sviluppate manualmente, che sostituiscono le *stub* del modello e mantengono separati il livello decisionale e il livello di integrazione board-specific.

## 13.1 Driver principali

### 13.1.1 Driver della Board1

#### **motor**

Il driver `motor` astrae il pilotaggio dei quattro attuatori tramite PWM e direzione. Gestisce l'inizializzazione dei canali timer e la conversione del comando di velocità in duty-cycle. Include vincoli di saturazione e funzioni di arresto rapido, utili nelle routine di sicurezza. Espone un'interfaccia uniforme verso la logica di controllo, evitando accessi diretti ai registri HAL.

#### **encoder**

Il driver `encoder` acquisisce la velocità ruota a partire dai conteggi incrementali. Si occupa della lettura periodica dei timer in modalità encoder e del calcolo dei delta campionati. Applica la conversione in RPM secondo la configurazione meccanica (CPR e rapporto di riduzione). Fornisce misure coerenti ai task PID, riducendo il rumore con logiche di validazione temporale.

#### **batt**

Il driver `batt` legge la tensione di alimentazione attraverso ADC e rete di partizione. Gestisce la calibrazione del canale e la trasformazione da valore digitale a tensione reale. Le misure vengono confrontate con soglie operative per prevenire condizioni di insufficienza energetica.

#### **temp**

Il driver `temp` misura la temperatura di sistema e la rende disponibile al supervisionamento. Integra la catena di acquisizione analogica e la linearizzazione del sensore utilizzato. Le misure vengono confrontate con soglie operative per prevenire condizioni di surriscaldamento.

### **led / led\_stripes**

I driver `led` e `led_stripes` implementano la segnalazione visiva di stato del rover. Gestiscono pattern differenti con timing dedicato.

## **13.1.2 Driver della Board2**

### **hcsr04**

Il driver `hcsr04` gestisce i tre sonar ultrasonici per la percezione di prossimità. Coordina trigger, cattura eco e conversione tempo-distanza in modo non bloccante. I dati risultanti alimentano la selezione delle routine evasive nella logica decisionale.

### **mpu6050**

Il driver `mpu6050` acquisisce i dati inerziali necessari alla stima dell'assetto locale. Configura il dispositivo su bus I2C, legge registri sensore e gestisce codici di errore di comunicazione. Rende disponibile in particolare l'informazione di yaw usata nelle manovre di rotazione.

### **controller**

Il driver `controller` raccoglie gli input utente (stick, pulsanti, stato batteria telecomando). Si occupa dell'acquisizione e della normalizzazione dei comandi. Introduce controlli di coerenza e dead-zone per migliorare stabilità e guidabilità del rover.

### **motor (open-loop)**

Su Board2 il driver `motor` è identico a quello della Board1, tuttavia a livello superiore viene usato in modalità open-loop per applicare comandi diretti.

### **dwt\_delay**

Il driver `dwt_delay` fornisce ritardi a granularità fine basati sul contatore DWT Cortex-M. Viene impiegato dove sono richieste temporizzazioni precise senza dipendere dal tick RTOS, migliorando accuratezza temporale locale.

## 13.2 Repository

Gli aspetti implementativi di dettaglio sono documentati nel codice sorgente ufficiale del progetto, consultabile al seguente repository GitHub: [https://github.com/ecav-25/cric\\_rover.git](https://github.com/ecav-25/cric_rover.git) La struttura del repository è organizzata in modo speculare per le due board, così da mantenere coerenza architetturale e facilitare la navigazione del codice.

### 13.2.1 board1\_firmware

```

1 board1_firmware/
2   `-- Core/
3     |-- Inc/
4     `-- Src/
5       |-- main.c
6       |-- app_freertos.c
7       |-- Board1.c
8       |-- board1_functions.c
9       |-- motor.c
10      |-- motor_diagnostic.c
11      |-- pid.c
12      |-- pid_law.c
13      |-- encoder.c
14      |-- batt.c
15      |-- temp.c
16      |-- led.c
17      |-- led_stripes.c
18      |-- hw_config.c
19      `-- deadline_watchdog.c

```

### 13.2.2 board2\_firmware

```

1 board2_firmware/
2   `-- Core/
3     |-- Inc/
4     `-- Src/
5       |-- main.c
6       |-- app_freertos.c
7       |-- Board2.c
8       |-- board2_functions.c
9       |-- controller.c
10      |-- hcsr04.c
11      |-- mpu6050.c
12      |-- motor.c
13      |-- dwt_delay.c
14      |-- hw_config.c
15      `-- deadline_watchdog.c

```

---

# Test di funzionamento

---

In questo capitolo sono descritti i test di integrazione e le prove di funzionamento eseguite direttamente sul sistema fisico, al fine di verificare il corretto comportamento complessivo dell'hardware e del firmware sviluppato. Le attività riportate riguardano la validazione dell'interazione tra le diverse componenti del sistema e la verifica operativa delle funzionalità implementate. Per quanto concerne i test unitari delle singole funzioni e dei moduli software, si rimanda direttamente ai codici sorgenti, nei quali sono documentate le relative procedure di verifica.

## 14.1 Test di movimento

<b>ID</b>	T-RM-1
<b>Precondizione</b>	<ul style="list-style-type: none"><li>• Rover acceso.</li><li>• Board 2 operativa</li><li>• Controller remoto connesso e funzionante.</li><li>• Levetta di sterzata in posizione neutra.</li><li>• Nessuna frenata attiva.</li><li>• Nessuna condizione critica di sicurezza è attiva (batteria, temperatura, panic, errori su motori o sensori).</li><li>• I sensori a ultrasuoni anteriori non rilevano ostacoli nella direzione di marcia tali da inibire l'avanzamento.</li></ul>
<b>Azione</b>	L'utente inclina in avanti la levetta dedicata al controllo della velocità.
<b>Output Atteso</b>	Il rover inizia a traslare in avanti lungo una traiettoria rettilinea.
<b>Postcondizione</b>	<ul style="list-style-type: none"><li>• I riferimenti di velocità delle quattro ruote risultano uguali e positivi.</li><li>• Il precedente riferimento di comando ai motori è sostituito dal nuovo riferimento di avanzamento.</li></ul>

Tabella 14.1: T-RM-1: Traslazione in avanti

<b>ID</b>	T-RM-2
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Board 2 operativa</li> <li>• Controller remoto connesso e funzionante.</li> <li>• Retromarcia speciale abilitata.</li> <li>• Levetta di sterzata in posizione neutra.</li> <li>• Nessuna frenata attiva.</li> <li>• Nessuna condizione di sicurezza attiva (stato critico della batteria, temperatura elevata, combinazione di emergenza, errori sui motori o blocchi di sicurezza).</li> </ul>
<b>Azione</b>	L'utente inclina all'indietro la levetta dedicata al controllo della velocità.
<b>Output Atteso</b>	Il rover inizia a traslare all'indietro lungo una traiettoria rettilinea.
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>• I riferimenti di velocità delle quattro ruote risultano uguali e negativi.</li> <li>• Il precedente riferimento di comando ai motori è sostituito dal nuovo riferimento di retromarcia.</li> </ul>

Tabella 14.2: T-RM-2: Traslazione indietro

<b>ID</b>	T-RM-3
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Board 2 operativa</li> <li>• Controller remoto connesso e funzionante.</li> <li>• Retromarcia speciale abilitata.</li> <li>• Nessuna frenata attiva.</li> <li>• Nessuna condizione critica attiva (batteria critica, temperatura elevata, panic combo, errori motore).</li> <li>• I sensori a ultrasuoni anteriori non rilevano ostacoli nella direzione di marcia tali da inibire l'avanzamento.</li> </ul>
<b>Azione</b>	L'utente inclina la levetta di controllo della velocità in avanti o all'indietro e, contemporaneamente, inclina la levetta di sterzata lateralmente.
<b>Output Atteso</b>	Il rover si muove descrivendo una traiettoria curva, combinando una componente di traslazione (avanti o indietro) e una componente di rotazione.

Tabella 14.3: T-RM-3: Traslazione con rotazione (traiettoria curva)

### 14.1.1 T-RM-4 — Rotazione sul posto

ID	T-RM-4
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso e fermo.</li> <li>Board 2 operativa</li> <li>Controller remoto connesso e funzionante.</li> <li>Nessuna frenata attiva.</li> <li>Levetta di controllo della velocità in posizione neutra.</li> <li>Nessuna condizione critica attiva (batteria critica, temperatura elevata, panic combo, errori motore).</li> </ul>
Azione	L'utente inclina lateralmente la levetta di sterzata.
Output Atteso	Il rover ruota sul posto attorno al proprio baricentro, senza componente di traslazione longitudinale.
Postcondizione	<ul style="list-style-type: none"> <li>I riferimenti di velocità delle ruote risultano uguali in modulo e opposti in segno sui due lati del rover.</li> <li>Il riferimento di comando precedente ai motori è sostituito da un riferimento di rotazione pura.</li> </ul>

Tabella 14.4: T-RM-4: Rotazione sul posto

### 14.1.2 T-RM-5 — Retromarcia speciale con rotazione di 180°

ID	T-RM-5
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Board 2 operativa</li> <li>Controller remoto connesso e funzionante.</li> <li>Retromarcia speciale abilitata.</li> <li>Levetta di controllo della velocità inclinata all'indietro.</li> <li>Nessuna condizione critica attiva (batteria critica, temperatura elevata, panic combo, errori motore).</li> </ul>
Azione	L'utente comanda la retromarcia con funzionalità speciale attiva.
Output Atteso	Il rover esegue una manovra di rotazione di 180° sul posto e, al termine della rotazione.
Postcondizione	<ul style="list-style-type: none"> <li>L'orientamento del rover risulta invertito di 180° rispetto a quello iniziale.</li> <li>La modalità di retro speciale resta abilitata.</li> </ul>

Tabella 14.5: T-RM-5: Retromarcia speciale con rotazione di 180°

### 14.1.3 T-RM-6 — Frenata progressiva (smooth)

<b>ID</b>	T-RM-6
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Board 2 operativa</li> <li>• Rover in movimento (avanti, indietro o lungo traiettoria curva).</li> <li>• Nessuna condizione critica attiva (batteria critica, temperatura elevata, panic combo, errori motore).</li> </ul>
<b>Azione</b>	L'utente attiva il comando di frenata smooth.
<b>Output Atteso</b>	La velocità del rover diminuisce in modo continuo e progressivo, senza variazioni brusche, fino all'arresto completo.
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>• I riferimenti di velocità delle ruote convergono a zero.</li> <li>• La modalità di frenata risulta impostata su stato smooth.</li> <li>• Il rover risulta fermo indipendentemente dal comando di marcia precedentemente attivo.</li> </ul>

Tabella 14.6: T-RM-6: Frenata progressiva (smooth)

### 14.1.4 T-RM-7 — Frenata di emergenza

<b>ID</b>	T-RM-7
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Board 2 operativa</li> <li>• Controller remoto connesso e funzionante.</li> <li>• Rover in movimento (avanti, indietro o lungo traiettoria curva).</li> </ul>
<b>Azione</b>	L'utente attiva il comando di frenata di emergenza.
<b>Output Atteso</b>	Il rover interrompe istantaneamente la marcia, annullando ogni riferimento di velocità.
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>• I riferimenti di velocità delle quattro ruote sono forzati a zero.</li> <li>• La modalità di frenata risulta impostata su stato di emergenza.</li> <li>• Il rover risulta fermo indipendentemente dal comando di marcia precedentemente attivo.</li> </ul>

Tabella 14.7: T-RM-7: Frenata di emergenza

## 14.2 Controllo

### 14.2.1 T-RCO-1 — Controllo PID

<b>ID</b>	T-RCO-1
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Entrambe le board operative e comunicanti</li> <li>• Nessuna condizione critica attiva(batteria critica, temperatura elevata, panic combo, errori motore).</li> </ul>
<b>Azione</b>	Il sistema riceve un comando di marcia dal controller remoto.
<b>Output Atteso</b>	Il controllo dei motori avviene in retroazione tramite un controllo PID.
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>• Il pid aggiorna la propria uscita di controllo per minimizzare l'errore in base al feedback degli encoder.</li> </ul>

Tabella 14.8: T-RCO-1: Controllo PID

### 14.2.2 T-RCO-2 — Controllo open-loop

<b>ID</b>	T-RCO-2
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Entrambe le board operative e comunicanti</li> <li>• Nessuna condizione critica attiva(batteria critica, temperatura elevata, panic combo, errori motore).</li> </ul>
<b>Azione</b>	Il sistema rileva un errore di comunicazione tra le due Board.
<b>Output Atteso</b>	Il sistema disabilita il controllo in retroazione e passa al controllo open-loop.
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>• Il sistema entra in stato operativo degradato/single board.</li> <li>• I riferimenti di velocità sono applicati direttamente agli attuatori senza feedback dagli encoder.</li> </ul>

Tabella 14.9: T-RCO-2: Passaggio a controllo open-loop in caso di fault

## 14.3 Attuazione delle luci

### 14.3.1 T-RAL-1 — Selezione modalità luci

ID	T-RAL-1
Precondizione	<ul style="list-style-type: none"><li>• Rover acceso.</li><li>• Entrambe le board operative e comunicanti</li><li>• Controller remoto connesso e funzionante.</li></ul>
Azione	L'utente preme ripetutamente il pulsante associato alla levetta analogica destra.
Output Atteso	La modalità di illuminazione avanza ciclicamente secondo la sequenza OFF → ON → AUTO.
Postcondizione	La modalità di illuminazione attiva si adatta coerentemente con il numero di pressioni effettuate.

Tabella 14.10: T-RAL-1: Selezione delle modalità di illuminazione

### 14.3.2 T-RAL-2 — Modalità OFF

ID	T-RAL-2
Precondizione	<ul style="list-style-type: none"><li>• Rover acceso.</li><li>• Entrambe le board operative e comunicanti</li><li>• Modalità di illuminazione impostata su OFF.</li></ul>
Azione	Il rover rimane fermo o si muove in qualsiasi direzione.
Output Atteso	Tutte le luci del rover risultano spente.
Postcondizione	Il sistema permane nello stato di luci OFF.

Tabella 14.11: T-RAL-2: Modalità OFF

### 14.3.3 T-RAL-3 — Modalità ON

ID	T-RAL-3
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Modalità di illuminazione impostata su ON.</li> </ul>
Azione	Il rover rimane fermo oppure si muove in avanti o all'indietro.
Output Atteso	<ul style="list-style-type: none"> <li>Le luci frontali sono accese di colore bianco.</li> <li>Il comportamento delle luci posteriori dipende dallo stato di marcia: <ul style="list-style-type: none"> <li><b>Rover fermo o in marcia avanti:</b> la striscia di LED posteriore visualizza una luce di posizione rossa.</li> <li><b>Rover in marcia indietro:</b> l'intera striscia di LED posteriore è accesa di colore bianco.</li> </ul> </li> </ul>
Postcondizione	Il sistema permane nello stato di illuminazione ON.

Tabella 14.12: T-RAL-3: Modalità ON

### 14.3.4 T-RAL-4 Modalità AUTO

ID	T-RAL-4
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Modalità di illuminazione impostata su AUTO.</li> </ul>
Azione	Il rover è fermo.
Output Atteso	<ul style="list-style-type: none"> <li>Le luci frontali risultano spente.</li> <li>Le luci posteriori sono accese di colore rosso.</li> <li>La striscia di LED posteriore visualizza una luce di posizione rossa.</li> </ul>
Postcondizione	Il sistema permane nello stato di luci AUTO.

Tabella 14.13: T-RAL-4: Modalità AUTO

<b>ID</b>	T-RAL-5
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Modalità di illuminazione impostata su AUTO.</li> </ul>
<b>Azione</b>	Il rover si muove in avanti o all'indietro.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>Le luci frontali risultano spente.</li> <li>Il comportamento delle luci posteriori dipende dallo stato di marcia: <ul style="list-style-type: none"> <li><b>Rover fermo o in marcia avanti:</b> la striscia di LED posteriore visualizza una luce di posizione rossa.</li> <li><b>Rover in marcia indietro:</b> l'intera striscia di LED posteriore è accesa di colore bianco.</li> </ul> </li> </ul>
<b>Postcondizione</b>	Il sistema permane nello stato di illuminazione AUTO.

Tabella 14.14: T-RAL-4: Modalità AUTO

#### 14.3.5 T-RAL-6 — Traslazione laterale (sinistra / destra)

<b>ID</b>	T-RAL-6
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Modalità di illuminazione diversa da OFF.</li> <li>Rover in movimento con componente di sterzata attiva.</li> </ul>
<b>Azione</b>	Il rover trasla lateralmente verso sinistra oppure verso destra.
<b>Output Atteso</b>	<p>Il comportamento delle luci dipende dalla direzione di traslazione laterale:</p> <ul style="list-style-type: none"> <li><b>Traslazione verso sinistra:</b> <ul style="list-style-type: none"> <li>La luce frontale sinistra lampeggi di colore rosso.</li> <li>La luce frontale destra è accesa di colore bianco.</li> <li>Le luci posteriori simulano un'animazione che riempie progressivamente metà dei LED verso sinistra.</li> </ul> </li> <li><b>Traslazione verso destra:</b> <ul style="list-style-type: none"> <li>La luce frontale destra lampeggi di colore rosso.</li> <li>La luce frontale sinistra è accesa di colore bianco.</li> <li>Le luci posteriori simulano un'animazione che riempie progressivamente metà dei LED verso destra.</li> </ul> </li> </ul>
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>Il sistema mantiene la configurazione luminosa coerente con la direzione di traslazione laterale finché persiste la manovra.</li> <li>Il sistema permane nello stato di illuminazione precedentemente impostato.</li> </ul>

Tabella 14.15: T-RAL-6: Traslazione laterale (sinistra / destra)

#### 14.3.6 T-RAL-7 — Retromarcia speciale

ID	T-RAL-7
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Modalità di retromarcia speciale attiva.</li> </ul>
Azione	Il rover esegue una manovra di retromarcia speciale.
Output Atteso	<p>Il comportamento delle luci dipende dalla modalità di illuminazione selezionata:</p> <ul style="list-style-type: none"> <li><b>Modalità ON o AUTO:</b> <ul style="list-style-type: none"> <li>La striscia di LED posteriore visualizza un'animazione dinamica multicolore (arcobaleno).</li> <li>Le due luci frontali lampeggiano di colore bianco.</li> </ul> </li> <li><b>Modalità OFF:</b> <ul style="list-style-type: none"> <li>Tutte le luci risultano spente.</li> </ul> </li> </ul>
Postcondizione	Il sistema mantiene la configurazione luminosa coerente con la retromarcia speciale finché la manovra è attiva.

Tabella 14.16: T-RAL-7: Segnalazione luminosa durante la retromarcia speciale

#### 14.3.7 T-RAL-8 — Segnalazione luminosa modalità

ID	T-RAL-8
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Nessuna condizione operativa anomala attiva.</li> </ul>
Azione	L'utente seleziona una modalità di controllo della dinamica di marcia tra <b>DEFAULT</b> , <b>ECO</b> e <b>SPORT</b> .
Output Atteso	<p>La targhetta posteriore assume un colore univocamente associato alla modalità selezionata:</p> <ul style="list-style-type: none"> <li><b>DEFAULT</b>: colore viola.</li> <li><b>ECO</b>: colore verde.</li> <li><b>SPORT</b>: colore arancione.</li> </ul>
Postcondizione	La targhetta mantiene il colore associato alla modalità di dinamica di marcia attiva.

Tabella 14.17: T-RAL-8: Segnalazione della dinamica di marcia

<b>ID</b>	T-RAL-9
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> </ul>
<b>Azione</b>	Il sistema entra in una modalità di funzionamento degradato.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>La targhetta posteriore diventa gialla.</li> <li>•</li> </ul>
<b>Postcondizione</b>	La segnalazione di funzionamento degradato permane finché il sistema non rientra nello stato operativo normale o non evolve in una condizione operativa anomala.

Tabella 14.18: T-RAL-9: Segnalazione di funzionamento degradato

<b>ID</b>	T-RAL-10
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti.</li> </ul>
<b>Azione</b>	Si verifica un errore grave nella comunicazione come disallineamento nello stato del sistema o nella decisione.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>La targhetta posteriore diventa rossa.</li> <li>Le luci anteriori lampeggiano di colore rosso.</li> <li>La striscia posteriore si colora di rosso.</li> </ul>
<b>Postcondizione</b>	La segnalazione luminosa permane, una diversa segnalazione luminosa è possibile solo mediante il reset completo del sistema.

Tabella 14.19: T-RAL-10: Segnalazione modalità Single-Board

<b>ID</b>	T-RAL-11
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> </ul>
<b>Azione</b>	Viene attivata la panic combo tramite la sequenza prevista dei pulsanti.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>La targhetta posteriore diventa rossa.</li> <li>Le luci anteriori lampeggiano di colore rosso.</li> <li>La striscia posteriore si colora di rosso.</li> </ul>
<b>Postcondizione</b>	La segnalazione luminosa permane, una diversa segnalazione luminosa è possibile solo mediante il reset completo del sistema.

Tabella 14.20: T-RAL-11: Segnalazione attivazione Panic Combo

<b>ID</b>	T-RAL-12
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti.</li> </ul>
<b>Azione</b>	Si verifica almeno una delle seguenti condizioni: <ul style="list-style-type: none"> <li>Errore critico su motori.</li> <li>Errore critico su encoder.</li> </ul>
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>La targhetta posteriore diventa rossa.</li> <li>Le luci anteriori lampeggiano di colore rosso.</li> <li>La striscia posteriore si colora di rosso.</li> </ul>
<b>Postcondizione</b>	La segnalazione luminosa permane, una diversa segnalazione luminosa è possibile solo mediante il reset completo del sistema.

Tabella 14.21: T-RAL-12: Segnalazione errore critico motori o encoder

## 14.4 Controllo della dinamica di marcia

### 14.4.1 T-RCM-1 — Selezione della modalità di dinamica di marcia

<b>ID</b>	T-RCM-1
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti.</li> <li>Modalità di dinamica di marcia inizialmente impostata su <b>DEFAULT</b>.</li> </ul>
<b>Azione</b>	L'utente preme ripetutamente il pulsante associato alla levetta analogica sinistra.
<b>Output Atteso</b>	La modalità di dinamica di marcia varia ciclicamente secondo la sequenza: <b>DEFAULT</b> → <b>SPORT</b> → <b>ECO</b> .
<b>Postcondizione</b>	La modalità di dinamica di marcia attiva corrisponde all'ultima selezione effettuata dall'utente.

Tabella 14.22: T-RCM-1: Selezione ciclica della modalità di dinamica di marcia

### 14.4.2 T-RCM-2 — Modalità di default all'accensione

<b>ID</b>	T-RCM-2
<b>Precondizione</b>	Rover spento.
<b>Azione</b>	Il rover viene acceso.
<b>Output Atteso</b>	La modalità di dinamica di marcia risulta impostata su <b>DEFAULT</b> .
<b>Postcondizione</b>	Il sistema opera in modalità <b>DEFAULT</b> fino a una successiva selezione dell'utente.

Tabella 14.23: T-RCM-2: Modalità di dinamica di marcia di default

### 14.4.3 T-RCM-3 — Influenza della modalità sulla dinamica di variazione della velocità

ID	T-RCM-3
Precondizione	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti</li> <li>Rover inizialmente fermo.</li> </ul>
Azione	<p>A parità di comando di accelerazione:</p> <ul style="list-style-type: none"> <li>l'utente seleziona la modalità <b>DEFAULT</b>;</li> <li>successivamente <b>ECO</b>;</li> <li>successivamente <b>SPORT</b>.</li> </ul>
Output Atteso	<p>La variazione della velocità del rover presenta una dinamica osservabile coerente con la modalità selezionata:</p> <ul style="list-style-type: none"> <li><b>DEFAULT</b>: variazione standard.</li> <li><b>ECO</b>: variazione più graduale e conservativa.</li> <li><b>SPORT</b>: variazione più rapida e reattiva.</li> </ul>
Postcondizione	Il rover continua a rispondere ai comandi di velocità secondo la dinamica della modalità attiva.

Tabella 14.24: T-RCM-3: Influenza della modalità sulla dinamica di marcia

## 14.5 Limitazione della velocità

### 14.5.1 T-RLM-1 — Limiti di velocità di default

ID	T-RLM-1
Precondizione	Rover spento.
Azione	Il rover viene acceso.
Output Atteso	I limiti minimo e massimo di velocità risultano impostati rispettivamente a 50 rpm e 150 rpm.
Postcondizione	I limiti di velocità vengono mantenuti fintanto che l'utente non li modifichi manualmente o si entri in qualche modalità che limiti automaticamente la velocità.

Tabella 14.25: T-RLM-1: Limiti di velocità di default

### 14.5.2 T-RLM-2 — Regolazione dei limiti di velocità

<b>ID</b>	T-RLM-2
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le board operative e comunicanti.</li> <li>Limiti di velocità inizialmente impostati ai valori di default.</li> </ul>
<b>Azione</b>	<ul style="list-style-type: none"> <li>L'utente incrementa progressivamente la velocità massima tramite la combinazione di comando dedicata.</li> <li>L'utente decrementa progressivamente la velocità massima tramite la combinazione di comando dedicata.</li> <li>L'utente riduce la velocità massima al di sotto della velocità di marcia corrente.</li> </ul>
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>La velocità massima consentita varia progressivamente entro i limiti minimo e massimo ammessi dal sistema.</li> <li>Quando il nuovo limite massimo è inferiore alla velocità di marcia corrente, il rover riduce la velocità in modo controllato fino a rientrare nel limite.</li> </ul>
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>Il rover opera nel rispetto dei limiti di velocità attualmente configurati.</li> <li>I limiti di velocità vengono mantenuti fintanto che l'utente non li modifichi manualmente o si entri in qualche modalità che limiti automaticamente la velocità.</li> </ul>

Tabella 14.26: T-RLM-2: Regolazione dei limiti di velocità

## 14.6 Rilevamento degli ostacoli

### 14.6.1 T-RRO-1 — Configurazione e portata dei sensori

<b>ID</b>	T-RRO-1
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso</li> <li>Entrambe le board operative e comunicanti</li> </ul>
<b>Azione</b>	Il rover acquisisce misure dai tre sensori a ultrasuoni.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>Tutti i sensori rilevano ostacoli fino ad almeno 3 m.</li> </ul>
<b>Postcondizione</b>	Il sistema utilizza correttamente le informazioni di distanza per il rilevamento ostacoli.

Tabella 14.27: T-RRO-1: Configurazione sensori a ultrasuoni

#### 14.6.2 T-RRO-2 — Modalità di sensibilità

<b>ID</b>	T-RRO-2
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso</li> <li>• Entrambe le board operative e comunicanti</li> </ul>
<b>Azione</b>	L'utente attiva la combinazione di comando per il cambio modalità di sensibilità.
<b>Output Atteso</b>	Il sistema alterna correttamente tra modalità <b>prudente</b> e <b>rilassata</b> .
<b>Postcondizione</b>	La modalità selezionata resta attiva fino a una nuova selezione o fino ad un'alterazione della modalità operativa Normale.

Tabella 14.28: T-RRO-2: Selezione modalità di sensibilità

#### 14.6.3 T-RRO-3 — Ostacolo frontale progressivo (prudente)

<b>ID</b>	T-RRO-3
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover in marcia avanti</li> <li>• Modalità di sensibilità <b>prudente</b>.</li> </ul>
<b>Azione</b>	Un ostacolo viene rilevato frontalmente a distanza compresa tra 70 cm e 3 m.
<b>Output Atteso</b>	Il rover riduce la velocità e arresta la marcia con dinamica smooth.
<b>Postcondizione</b>	Il rover rimane fermo in sicurezza.

Tabella 14.29: T-RRO-3: Ostacolo frontale progressivo

#### 14.6.4 T-RRO-4 — Ostacolo improvviso ravvicinato (prudente)

<b>ID</b>	T-RRO-4
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover in marcia</li> <li>• Modalità di sensibilità <b>prudente</b></li> </ul>
<b>Azione</b>	Un ostacolo improvviso viene rilevato da uno qualunque dei sensori a distanza inferiore a 70 cm.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>• Il rover esegue una frenata di emergenza</li> <li>• Se possibile, effettua una manovra evasiva coerente con la disponibilità laterale.</li> </ul>
<b>Postcondizione</b>	Il rover risulta fermo oppure orientato in direzione sicura.

Tabella 14.30: T-RRO-4: Ostacolo improvviso ravvicinato

#### 14.6.5 T-RRO-5 — Ostacolo laterale anticipato (prudente)

ID	T-RRO-5
Precondizione	<ul style="list-style-type: none"> <li>Rover in marcia</li> <li>Modalità di sensibilità <b>prudente</b>.</li> </ul>
Azione	Un ostacolo viene rilevato inizialmente da uno o entrambi i sensori laterali a una distanza compresa tra 1.5 m e 3 m e, successivamente, viene rilevato dal sensore frontale alla stessa distanza entro 1000 ms.
Output Atteso	<p>Il comportamento del rover dipende dalla combinazione di rilevamenti:</p> <ul style="list-style-type: none"> <li>Se l'ostacolo è rilevato inizialmente da un solo sensore laterale, il rover modifica la traiettoria verso questo lato senza interrompere la marcia.</li> <li>Se l'ostacolo è rilevato inizialmente da entrambi i sensori laterali, il rover riduce la velocità ed esegue una frenata smooth.</li> </ul>
Postcondizione	La marcia prosegue oppure termina, coerentemente con l'azione intrapresa.

Tabella 14.31: T-RRO-5: Ostacolo laterale anticipato in modalità prudente

#### 14.6.6 T-RRO-6 — Ostacolo improvviso (rilassata)

ID	T-RRO-6
Precondizione	<ul style="list-style-type: none"> <li>Rover in marcia</li> <li>Modalità di sensibilità <b>rilassata</b>.</li> </ul>
Azione	Un ostacolo improvviso viene rilevato da uno o più sensori a distanza inferiore a 40 cm.
Output Atteso	<ul style="list-style-type: none"> <li>Il rover esegue una frenata di emergenza</li> <li>Se possibile, effettua una manovra evasiva coerente con la disponibilità laterale.</li> </ul>
Postcondizione	Il rover risulta fermo oppure orientato in direzione sicura.

Tabella 14.32: T-RRO-6: Ostacolo improvviso in modalità rilassata

## 14.7 Sicurezza

### 14.7.1 T-RFS-1 — Ingresso in modalità degradata

<b>ID</b>	T-RFS-1
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Entrambe le schede operative e comunicanti.</li> </ul>
<b>Azione</b>	Board 1 non risponde alla board 2 nella finestra prestabilita di comunicazione.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>Il controllo dei motori prosegue senza interruzioni visibili.</li> <li>I motori risultano controllati in modalità open-loop.</li> <li>La modalità viene segnalata mediante le opportune segnalazioni luminose.</li> </ul>
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>Il sistema entra in modalità di funzionamento degradata.</li> <li>Il controllo dei motori risulta affidato alla sola board 2.</li> <li>Il sistema può rientrare nello stato operativo normale qualora la comunicazione venga ripristinata.</li> </ul>

Tabella 14.33: T-RFS-1: Ingresso in modalità degradata

### 14.7.2 T-RFS-2 — Limitazione della velocità in modalità degradata

<b>ID</b>	T-RFS-2
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>Rover acceso.</li> <li>Sistema in modalità di funzionamento degradata.</li> </ul>
<b>Azione</b>	L'utente tenta di incrementare la velocità di marcia del rover.
<b>Output Atteso</b>	La velocità dei motori non supera il valore massimo di sicurezza previsto per la modalità degradata impostato a 80 rpm.
<b>Postcondizione</b>	Il rover continua a operare con velocità limitata finché permane la modalità degradata.

Tabella 14.34: T-RFS-2: Limitazione della velocità in modalità degradata

### 14.7.3 T-RFS-3 — Rilevamento ostacoli in modalità degradata

<b>ID</b>	T-RFS-3
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Sistema in modalità di funzionamento degradata.</li> <li>• Rover in movimento.</li> </ul>
<b>Azione</b>	Un ostacolo viene rilevato dai sensori a ultrasuoni.
<b>Output Atteso</b>	<p>Il rover arresta la marcia in presenza di:</p> <ul style="list-style-type: none"> <li>• un ostacolo rilevato frontalmente entro la distanza di sicurezza prevista di 3 m;</li> <li>• un ostacolo rilevato lateralmente entro la distanza di sicurezza prevista di 1.5 m.</li> </ul>
<b>Postcondizione</b>	Il rover permane fermo finché persiste finché non vengono rilevati ostacoli sotto la soglia di sicurezza o finché persiste la modalità degradata.

Tabella 14.35: T-RFS-3: Rilevamento ostacoli in modalità degradata

### 14.7.4 T-RFS-4 — Ingresso in modalità single-board

<b>ID</b>	T-RFS-4
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Entrambe le schede operative e comunicanti</li> </ul>
<b>Azione</b>	Il sistema rileva una condizione persistente di controllo discordante o di mancato allineamento decisionale tra le board.
<b>Output Atteso</b>	<ul style="list-style-type: none"> <li>• Il controllo dei motori viene affidato stabilmente alla sola board master.</li> <li>• I motori risultano fermi.</li> <li>• La modalità viene segnalata mediante le opportune segnalazioni luminose.</li> </ul>
<b>Postcondizione</b>	<ul style="list-style-type: none"> <li>• Il sistema entra in modalità <i>single-board</i> che è irreversibile.</li> <li>• Il ritorno a una modalità operativa diversa è possibile solo tramite reset completo del sistema.</li> </ul>

Tabella 14.36: T-RFS-4: Ingresso in modalità single-board

<b>ID</b>	T-RFS-5
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> </ul>
<b>Azione</b>	La board 2 risulta non funzionante.
<b>Output Atteso</b>	Il rover arresta immediatamente i motori.
<b>Postcondizione</b>	Il rover rimane in stato di arresto finché la condizione di errore non viene risolta.

Tabella 14.37: T-RFS-5: Arresto per malfunzionamento di entrambe le board

<b>ID</b>	T-RFS-6
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Entrambe le board operative e comunicanti</li> </ul>
<b>Azione</b>	Almeno uno dei due motor controller non risponde ai comandi.
<b>Output Atteso</b>	Il rover arresta i motori.
<b>Postcondizione</b>	Il ritorno a una modalità operativa diversa è possibile solo tramite reset completo del sistema.

Tabella 14.38: T-RFS-6: Arresto per mancata risposta di un motor controller

<b>ID</b>	T-RFS-7
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Entrambe le board operative e comunicanti</li> </ul>
<b>Azione</b>	Almeno uno degli encoder non funziona.
<b>Output Atteso</b>	Il rover arresta i motori.
<b>Postcondizione</b>	Il ritorno a una modalità operativa diversa è possibile solo tramite reset completo del sistema.

Tabella 14.39: T-RFS-7: Arresto per malfunzionamento di almeno un encoder

<b>ID</b>	T-RFS-8
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover acceso.</li> <li>• Entrambe le board operative e comunicanti.</li> </ul>
<b>Azione</b>	Il giroscopio non risponde alle richieste I2C.
<b>Output Atteso</b>	La rotazione viene completata utilizzando una modalità alternativa di esecuzione che non dipende dal giroscopio.
<b>Postcondizione</b>	Il sistema continua a operare in modalità di fallback finché la comunicazione I2C con il giroscopio non viene ripristinata.

Tabella 14.40: T-RFS-8: Completamento rotazione con giroscopio non disponibile

<b>ID</b>	T-RFS-9
<b>Precondizione</b>	<ul style="list-style-type: none"> <li>• Rover in movimento.</li> </ul>
<b>Azione</b>	Il sistema perde connessione con il controller.
<b>Output Atteso</b>	Il rover frena in maniera smooth.
<b>Postcondizione</b>	Il rover rimane fermo finché la connessione non viene ristabilita.

Tabella 14.41: T-RFS-9: Arresto per perdita connessione con controller

<b>ID</b>	T-RFS-10
<b>Precondizione</b>	<ul style="list-style-type: none"><li>• Rover acceso.</li><li>• Connessione con controller attiva.</li></ul>
<b>Azione</b>	La batteria del controller scende al di sotto del 5%.
<b>Output Atteso</b>	Il rover frena in maniera smooth.
<b>Postcondizione</b>	Il rover rimane fermo finché la condizione non viene risolta.

Tabella 14.42: T-RFS-10: Arresto per batteria controller &lt; 5%

<b>ID</b>	T-RFS-11
<b>Precondizione</b>	<ul style="list-style-type: none"><li>• Rover acceso.</li><li>• Comando di movimento attivo.</li></ul>
<b>Azione</b>	I motori non rispondono ai comandi impartiti.
<b>Output Atteso</b>	Il rover arresta i motori.
<b>Postcondizione</b>	Il ritorno a una modalità operativa diversa è possibile solo tramite reset completo del sistema.

Tabella 14.43: T-RFS-11: Arresto per mancata risposta dei motori

---

## Appendice A

### Printed Circuit Board

---

## A.1 PCB di controllo sicuro dei driver di potenza

### A.1.1 Problema di sicurezza nei driver Sabertooth

Durante le prime fasi di sperimentazione, l'integrazione dei motori driver Sabertooth ha evidenziato una criticità rilevante dal punto di vista della sicurezza operativa. I driver sono configurati in modalità analogica indipendente, in cui la velocità e la direzione dei motori sono determinate da un segnale PWM opportunamente filtrato, corrispondente a una tensione analogica compresa tra 1.875 V e 3.125 V.

In particolare, una tensione pari a 2.5 V corrisponde a velocità nulla, mentre scostamenti verso valori inferiori o superiori determinano rispettivamente una rotazione antioraria o oraria, con intensità proporzionale alla distanza dal punto di equilibrio. Tuttavia, il driver è privo di un pin diabilitazione dedicato. In assenza di un segnale definito, ovvero con il pin di controllo in stato *floating*, l'ADC interno del driver rileva una tensione pari a 0 V, interpretandola come un comando di rotazione antioraria alla massima potenza.

Questa condizione si manifesta sistematicamente in due scenari critici:

- durante la fase di accensione del sistema, poiché la generazione dei segnali di controllo avviene solo dopo il completamento del boot del sistema;
- in caso di guasto o arresto imprevisto della scheda di controllo, situazione in cui i motori inizierebbero a ruotare in modo incontrollato, generando potenziali condizioni di pericolo.

### A.1.2 Strategia di avvio sicuro dei driver

Per neutralizzare i rischi legati a rotazioni incontrollate dei motori, è stata adottata una strategia basata sul sezionamento fisico dell'alimentazione dei driver. In particolare, è stato introdotto un relè in configurazione normalmente aperta, che consente l'alimentazione dei driver esclusivamente quando il sistema di controllo ha raggiunto uno stato operativo stabile e sicuro.

### A.1.3 Selezione della sorgente di controllo

Seguendo le specifiche di progetto, il controllo dei motori deve essere possibile da entrambe le schede di controllo del rover, al fine di garantire un funzionamento degradato in caso di guasto, seppur privo di feedback sensoriale. Per questo motivo si è resa necessaria la realizzazione di un circuito in grado di selezionare dinamicamente la sorgente dei segnali di controllo.

La soluzione adottata è basata sull'utilizzo del circuito integrato 74HC157, un multiplexer digitale quad a due ingressi. L'integrato dispone di un segnale di selezione comune che consente di instradare, su ciascuna delle quattro uscite, una delle due sorgenti di controllo disponibili, come illustrato nello schema logico riportato in Figura A.1.

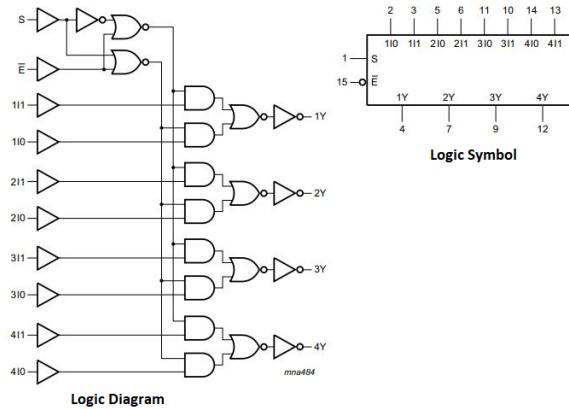
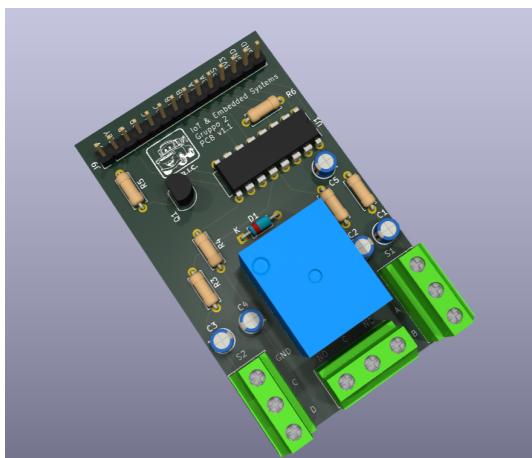


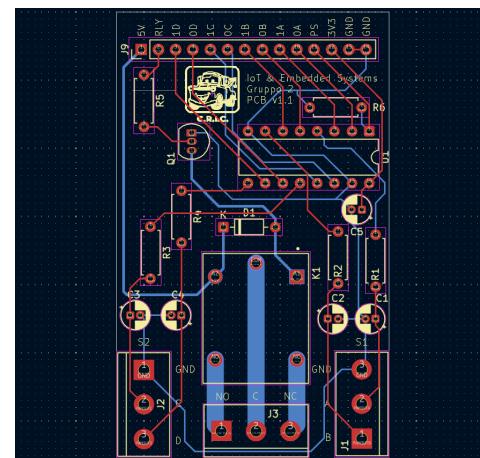
Figura A.1: Schema logico integrato 74HC157

#### A.1.4 Progettazione e realizzazione del circuito stampato

Per implementare in modo compatto e affidabile le soluzioni descritte, è stata progettata una PCB dedicata, ottimizzata per essere alloggiata in posizione centrale rispetto ai due driver di potenza. Oltre alla logica di selezione dei segnali, la scheda integra i quattro filtri RC necessari per la conversione dei segnali PWM in tensioni analogiche pulite, secondo le specifiche tecniche fornite dal produttore dei driver.



(a) Vista 3D della PCB di controllo motori



(b) Layout delle piste su PCB

Figura A.2: PCB di controllo motori: vista tridimensionale e layout delle piste

Lo schematico del circuito evidenzia l'integrazione del multiplexer 74HC157, dotato di resistenza di pull-up sul segnale di selezione e di un condensatore di decoupling, seguita dagli stadi di filtraggio RC e da uno stadio di potenza costituito da un relè SRD-05VDC-SL-C. Il relè è pilotato da un transistor NPN 2N3904, utilizzato come interruttore logico, ed è protetto da un diodo di flyback per la soppressione dei picchi di tensione generati dalla bobina.

### A.1.5 Interfaccia di collegamento e distribuzione dei segnali

L'interfaccia di collegamento tra la scheda di controllo motori e i sistemi di controllo del rover è stata progettata per raggruppare in modo ordinato le alimentazioni e i segnali

logici necessari al corretto funzionamento del sistema. In Figura A.3 è riportato lo schema complessivo dell'interfaccia di collegamento.

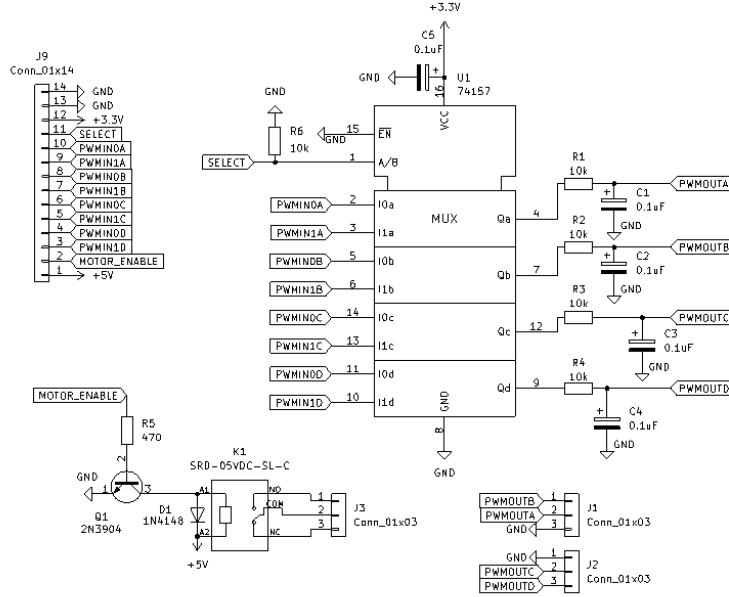


Figura A.3: Schema dell'interfaccia di collegamento e distribuzione dei segnali

L'interfaccia con i sistemi di controllo avviene tramite un pin header che raggruppa le alimentazioni e i segnali logici:

- Gestione Alimentazione:** Il circuito richiede una linea a 5V dedicata esclusivamente all'attivazione della bobina del relè e una linea a 3.3V per alimentare la logica interna e l'integrato di multiplexing. Il riferimento di massa (GND) è comune per tutti i segnali.
- Abilitazione Motori (RLY):** Il pin RLY accetta un segnale logico a 3.3V che agisce come interruttore di sicurezza, permettendo l'accensione o lo spegnimento dei driver di potenza solo quando il sistema è pronto.
- Segnali di Controllo (PIN <N><Y>):** La scheda riceve due set distinti di segnali PWM (Set 0 e Set 1). Ogni set contiene i comandi per i due canali (Y) necessari al movimento del veicolo.
- Selezione Sorgente (PS):** Il pin PWM Selection funge da selettore logico. Quando il segnale è alto, il multiplexer instrada alle uscite i comandi del Set 1; quando è basso, vengono inviati quelli del Set 0. Per garantire la stabilità in caso di disconnessione, un sistema di pull-down interno assicura che, se il pin resta "floating", il sistema selezioni di default il Set 0.

La disposizione fisica dei segnali sul pin header e la loro corrispondenza funzionale sono riportate in Figura A.4.

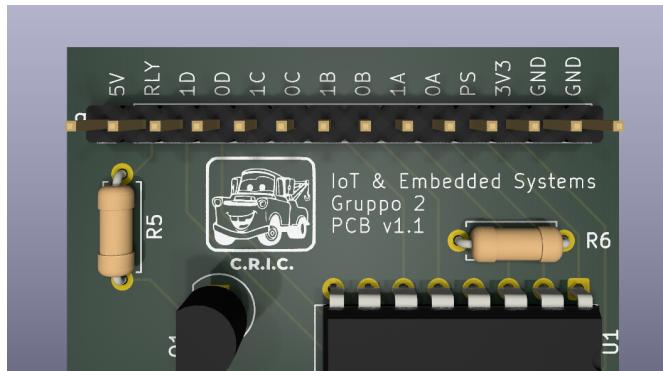


Figura A.4: Dettaglio del pin header di collegamento e distribuzione dei segnali

Il segnale elaborato e la potenza vengono distribuiti ai driver attraverso tre morsettiera a vite (Figura A.5).

Le uscite denominate A/B/C/D forniscono il segnale PWM opportunamente selezionato e filtrato per il controllo dei canali dei driver, affiancate dai relativi riferimenti di massa (GND). La sezione dedicata alla potenza gestisce i contatti del relè per il sezionamento elettrico. Questa include il contatto Comune (C) e le due uscite Normalmente Aperta (NO) e Normalmente Chiusa (NC). Nel progetto il contatto comune (C) è collegato alla linea di alimentazione in ingresso, mentre i driver sono connessi al contatto Normalmente Aperto (NO). In questa configurazione, in assenza di eccitazione della bobina del relè, l'alimentazione dei driver risulta interrotta; l'attivazione del segnale RLY consente la chiusura del contatto e l'alimentazione dei driver.

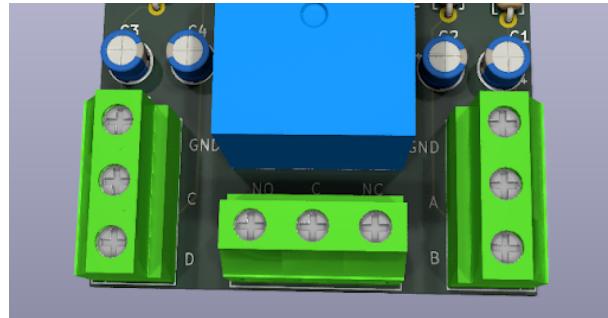


Figura A.5: Sezione di potenza e morsettiera di distribuzione dei segnali verso i driver

## A.2 PCB HAT per le board NUCLEO

Per agevolare e rendere più affidabile il cablaggio delle due board NUCLEO presenti nel sistema, sono state progettate e realizzate due PCB in formato *HAT*, in grado di sfruttare i connettori *Morpho* delle schede STMicroelectronics della famiglia NUCLEO-64, di cui la STM32G474 fa parte.

Le PCB HAT consentono di centralizzare le connessioni di alimentazione e di segnale, riducendo il cablaggio volante e migliorando l'affidabilità complessiva del sistema.

### A.2.1 PCB HAT della scheda Master

La scheda Master è dotata di una PCB HAT che permette il montaggio *on-board* delle periferiche I<sup>2</sup>C principali. In particolare, la scheda integra il sensore inerziale MPU6050, impiegato per la misura di accelerazioni e velocità angolari del veicolo.

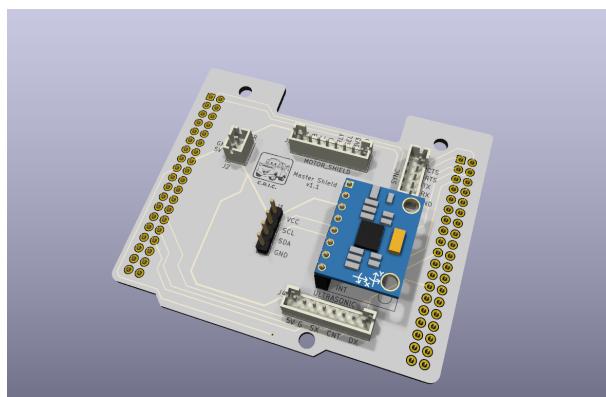


Figura A.6: PCB HAT della scheda Master

La PCB HAT della scheda Master dispone inoltre di morsettiera JST-PH da 2.0 mm per le seguenti connessioni:

- **POWER**, per l'alimentazione a 5 V;
- **ULTRASONIC**, per il collegamento dei tre sensori a ultrasuoni frontali HC-SR04;
- **MOTOR SHIELD**, per il collegamento alla PCB di controllo dei motori;
- **SYNC**, per la comunicazione con la scheda Slave.

### A.2.2 PCB HAT della scheda Slave

La PCB HAT realizzata per la scheda Slave è dedicata esclusivamente alla gestione delle connessioni di sistema e non prevede il montaggio diretto di periferiche *on-board*. Essa funge da nodo di interfaccia tra la logica di controllo, i sensori di feedback e gli attuatori del veicolo.

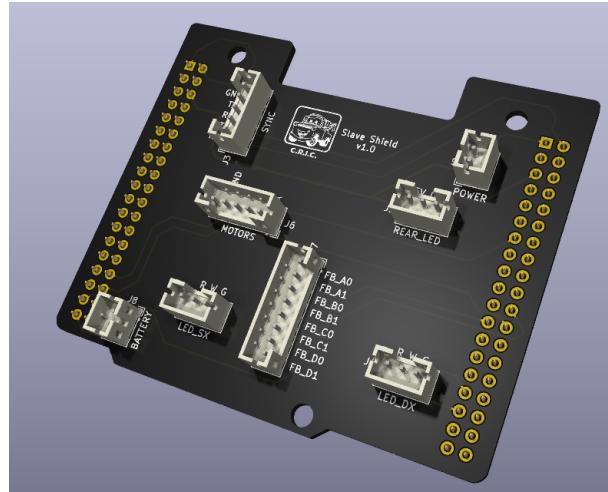


Figura A.7: PCB HAT della scheda Slave

La scheda integra esclusivamente connettori JST per le seguenti interfacce:

- **POWER**, per l'alimentazione a 5 V;
- **SYNC**, per la comunicazione con la scheda Master;
- **MOTORS**, per la connessione alla PCB di controllo dei motori;
- **FEEDBACK**, per la connessione degli encoder dei motori;
- **BATTERY**, per il monitoraggio della tensione della batteria del rover tramite partitore di tensione esterno;
- **LED\_DX** e **LED\_SX**, per il collegamento dei fari anteriori;
- **REAR\_LED**, per la connessione della strip LED posteriore WS2812.

Si osservi che la tensione di alimentazione a 12 V è presente esclusivamente nella sezione inferiore dello chassis e non viene distribuita direttamente alla logica di controllo.

---

## Appendice B

### Componenti meccanici stampati in 3D

---

Durante lo sviluppo del rover sono stati progettati e realizzati alcuni componenti meccanici tramite stampa 3D, al fine di supportare l'integrazione dell'elettronica di controllo e dei sensori all'interno della struttura del veicolo.

#### B.1 Controller del rover

È stato progettato e realizzato un contenitore stampato in 3D per il controller del rover, basato su un microcontrollore **ESP32** e dotato di un display centrale. Il case è stato concepito per alloggiare in modo ordinato e compatto i principali elementi di interazione con l'utente, garantendo al contempo robustezza meccanica e facilità di assemblaggio. Sulla parte frontale del contenitore sono presenti due pulsanti di comando e due stick analogici, utilizzati per il controllo del movimento del rover. Ulteriori due pulsanti sono collocati sulla parte frontale laterale del controller, in posizione facilmente accessibile durante l'utilizzo.

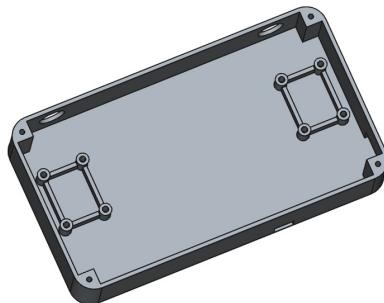


Figura B.1: Case controller

Al centro del pannello frontale è ricavata un'apertura dedicata al display, utilizzato per la visualizzazione delle informazioni di telemetria del sistema e dello stato operativo del rover.

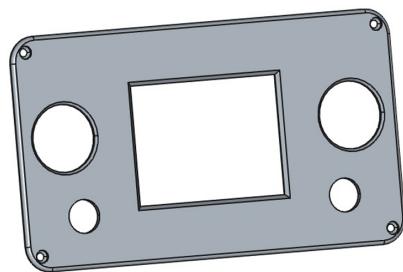


Figura B.2: Parte superiore del controller

## B.2 Case per le board di controllo STM32G4

Il componente mostrato in Figura B.3 è un supporto meccanico stampato in 3D progettato per l'alloggiamento e il fissaggio di una scheda di sviluppo **STM32G474RE**. Il supporto svolge la funzione di *holder* strutturale, consentendo il montaggio stabile della scheda sul rover consentendo di sfruttare i fori dedicati alle *Sabertooth* posizionate sul lato opposto del telaio. Il componente consente, inoltre, di mantenere la scheda sollevata rispetto al fondo del supporto, riducendo il rischio di contatti indesiderati e facilitando la gestione del cablaggio.



Figura B.3: Case di supporto per le STM32G4

La funzione principale del supporto è quella di facilitare il posizionamento stabile della scheda STM32 all'interno del sistema limitando movimenti indesiderati della board e riducendo i fenomeni di vibrazione e rimbalzo durante la guida del veicolo.

## B.3 Supporto per PCB di controllo dei motori

Il componente mostrato in Figura B.4 è un supporto meccanico stampato in 3D progettato per l'alloggiamento della PCB di controllo dei motori, descritta nel Paragrafo A.1. Il supporto consente di mantenere la scheda in posizione stabile e correttamente allineata all'interno del rover.

Poiché la PCB non è dotata di fori di fissaggio, è previsto che essa venga incollata direttamente all'interno della sede ricavata nel supporto. La cornice perimetrale garantisce il corretto posizionamento della scheda e contribuisce a limitarne i movimenti durante il funzionamento del veicolo.

Il fissaggio del supporto alla placca di montaggio del rover è ottenuto tramite fori compatibili con l'interasse standard delle schede *Single Board Computer Raspberry Pi*, semplificando l'integrazione meccanica con la struttura esistente.

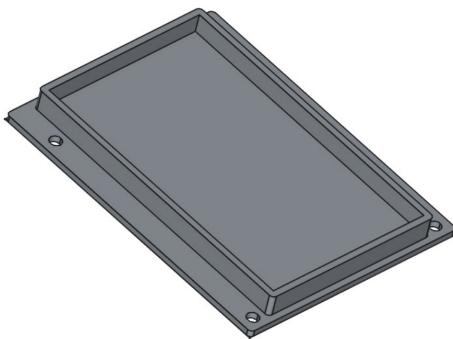


Figura B.4: Supporto stampato in 3D per la PCB di controllo dei motori

## B.4 Supporto per i sensori sonar

Il componente mostrato in Figura B.5 è un alloggiamento meccanico stampato in 3D progettato per ospitare un modulo di sensori a ultrasuoni **HC-SR04**, impiegato per il rilevamento di ostacoli e la percezione dell’ambiente da parte del rover.

L’alloggiamento è costituito da **due parti distinte**, una sezione anteriore e una sezione posteriore, stampate separatamente e progettate per essere assemblate tramite un sistema di *incastro meccanico*. Questa soluzione consente un montaggio semplice e rapido del sensore, senza l’utilizzo di viti aggiuntive, mantenendo al contempo una buona rigidità strutturale.

La parte anteriore del case presenta due aperture circolari di diametro adeguato, allineate con i trasduttori del modulo HC-SR04, in modo da non interferire con l’emissione e la ricezione delle onde ultrasoniche. Tra le due aperture è presente un’asola centrale che funge da riferimento geometrico e contribuisce alla corretta collocazione del modulo all’interno dell’alloggiamento.

La parte posteriore svolge la funzione di contenimento e supporto del corpo elettronico del sensore. Al suo interno sono presenti superfici di battuta che guidano il posizionamento del modulo e ne limitano i movimenti una volta completato l’incastro con la parte frontale. Sul retro è inoltre ricavato un **foro passante** dedicato al passaggio dei pin di collegamento e dei cavi, permettendo un instradamento ordinato delle connessioni elettriche verso l’interno del rover, riducendo al contempo sollecitazioni sui contatti.

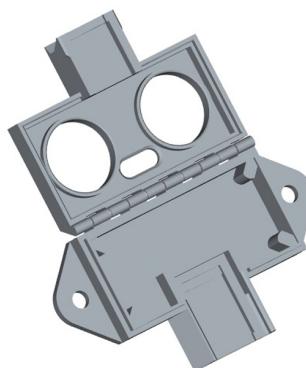


Figura B.5: Supporto per i sensori ad ultrasuoni

Nel complesso, il componente fornisce una soluzione compatta e funzionale per l'integrazione del sensore HC-SR04 nel sistema, combinando protezione meccanica, facilità di assemblaggio e corretta gestione dei collegamenti elettrici, contribuendo all'affidabilità del sistema di percezione a ultrasuoni durante il normale funzionamento del rover.

## B.5 Supporto per ESP32-CAM

Il supporto mostrato in Figura B.6 è un componente meccanico stampato in 3D progettato per l'alloggiamento del modulo **ESP-CAM**, installato nella parte anteriore del rover per la visione frontale. La geometria complessiva è studiata per garantire un fissaggio rigido al telaio e un corretto orientamento dell'obiettivo.

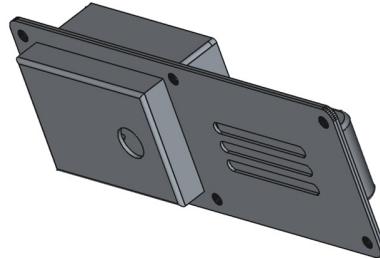


Figura B.6: Vista isometrica del supporto ESP-CAM assemblato

La piastra frontale, visibile in Figura B.7, integra l'apertura per l'obiettivo della camera e una serie di asole di ventilazione. Tali forature sono state introdotte per favorire il ricircolo dell'aria, necessario a smaltire il calore generato dal modulo durante il funzionamento.

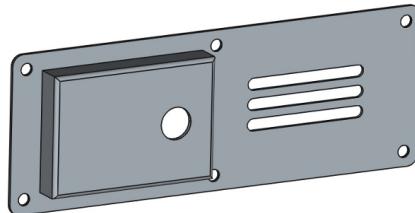


Figura B.7: Piastra frontale con apertura per l'obiettivo e asole di ventilazione

La Figura B.8 mostra il corpo posteriore del supporto, che funge da contenimento per il modulo ESP-CAM e per il dissipatore di calore. La forma interna è pensata per garantire un accoppiamento stabile del modulo, limitandone i movimenti durante la marcia del rover.

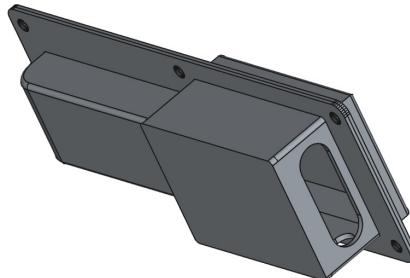


Figura B.8: Vista posteriore del supporto con sede per modulo e dissipatore

Infine, la vista interna riportata in Figura B.9 evidenzia la cavità di alloggiamento e il percorso dell'aria all'interno del supporto. Questa configurazione contribuisce a migliorare la dissipazione termica e ad aumentare l'affidabilità del modulo ESP-CAM durante il funzionamento continuo.

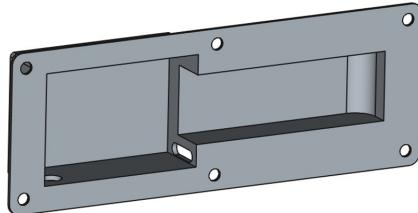


Figura B.9: Vista interna del supporto ESP-CAM

### B.5.1 Supporto per antenna esterna

Il componente mostrato in Figura B.10 è un supporto meccanico stampato in 3D progettato per il fissaggio di un'antenna esterna al rover. Il supporto è stato concepito per essere installato in corrispondenza di un angolo del telaio, così da minimizzare ingombri e interferenze con gli altri componenti.

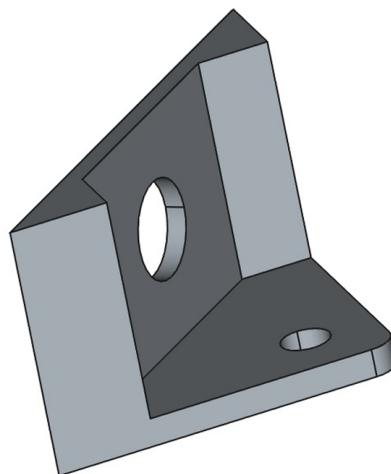


Figura B.10: Supporto stampato in 3D per antenna esterna

La geometria del pezzo presenta una configurazione angolare, con due superfici di appoggio ortogonali che consentono l'ancoraggio su due lati del rover. Le forature presenti sulle superfici di fissaggio sono posizionate in modo da permettere il montaggio diretto in prossimità dell'angolo, garantendo una connessione meccanica stabile e un corretto orientamento dell'antenna.

Il foro centrale sul piano verticale è dedicato all'alloggiamento del connettore dell'antenna, consentendo il passaggio del corpo radiante e dei cavi di collegamento verso l'interno del rover. Questa soluzione facilita l'instradamento dei cablaggi e contribuisce a mantenere ordinata l'integrazione dell'antenna nel sistema.

Nel complesso, il supporto fornisce una soluzione semplice e robusta per il montaggio dell'antenna esterna, assicurando stabilità meccanica e flessibilità di posizionamento sul telaio del rover.

## Appendice C

### Configurazione delle schede e schemi di collegamento

La presente appendice riporta la documentazione tecnica di dettaglio relativa alla configurazione hardware del sistema. Sono incluse la configurazione delle schede, le tabelle di assegnazione dei pin e gli schemi di collegamento adottati.

L'obiettivo è fornire una rappresentazione completa e verificabile delle scelte implementative a livello hardware, in coerenza con quanto descritto nei capitoli precedenti.

### C.1 Board 1 (Slave)

### C.1.1 Configurazione

La configurazione della prima scheda è stata definita tramite l'ambiente di configurazione STM32CubeMX. La figura seguente riporta l'assegnazione delle periferiche e dei pin utilizzati nel progetto.

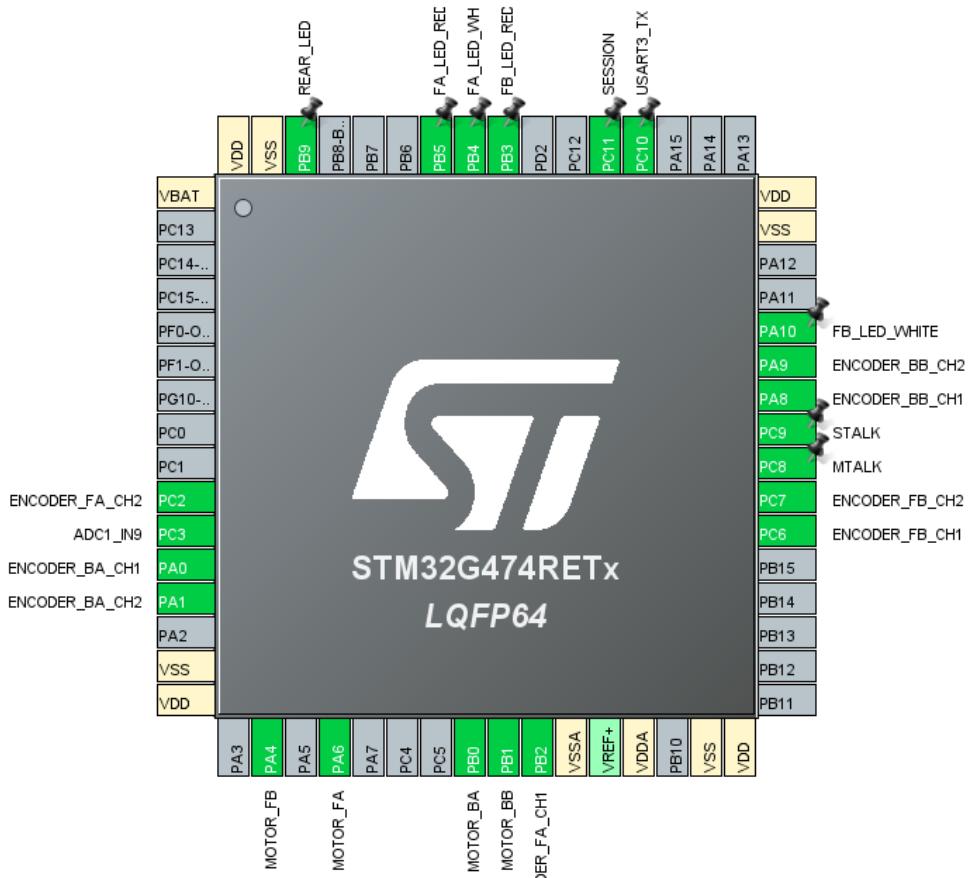


Figura C.1: Pinout della board Slave

### C.1.2 Tabella di assegnazione dei pin

La Tabella C.1 riporta in forma strutturata la corrispondenza tra i pin fisici e le funzionalità configurate.

Board Slave						
Component	Pinout					
	WHITE		RED			
Front LED FA	PB4		PB5			
Front LED FB	PA10		PB3			
COMMAND PIN						
Rear LEDs	TIM17_CH1 → PB9					
	TX	SESSION	MTALK	STALK		
Board Communication	USART3_TX → PC10	PC11	PC8	PC9		
ADC						
Analog Sensor	PC3 (ADC1_IN9)					
PWM						
MOTOR FA	TIM3_CH1 → PA6					
MOTOR FB	TIM3_CH2 → PA4					
MOTOR BA	TIM3_CH3 → PB0					
MOTOR BB	TIM3_CH4 → PB1					
	CHANNEL A		CHANNEL B			
Motor Encoder FA	TIM20_CH1 → PB2		TIM20_CH2 → PC2			
Motor Encoder FB	TIM8_CH1 → PC6		TIM8_CH2 → PC7			
Motor Encoder BA	TIM5_CH1 → PA0		TIM5_CH2 → PA1			
Motor Encoder BB	TIM1_CH1 → PA8		TIM1_CH2 → PA9			

Tabella C.1: Pinout Configuration for Board Slave

## C.2 Board 2 (Master)

### C.2.1 Configurazione

La configurazione della seconda scheda è stata definita in modo analogo, specificando le periferiche e i segnali necessari al corretto funzionamento del sistema.

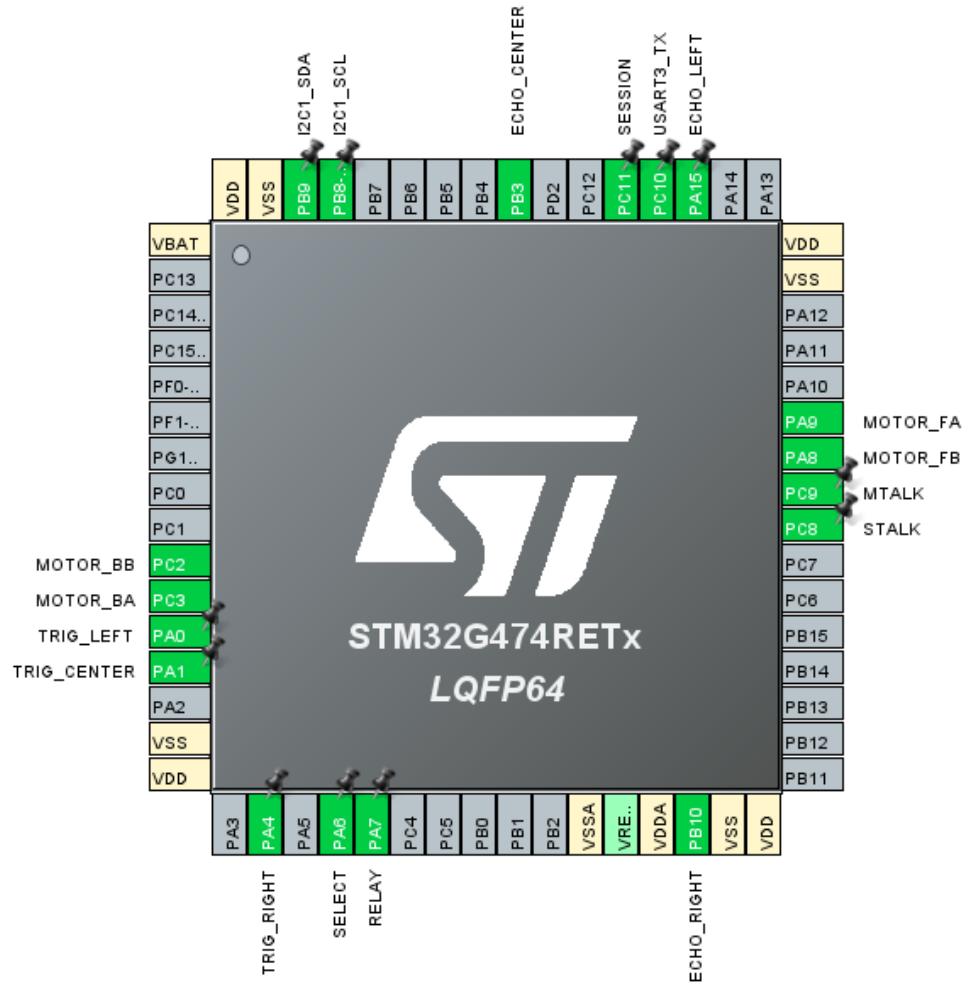


Figura C.2: Pinout della board Master

### C.2.2 Tabella di assegnazione dei pin

La Tabella C.2 riporta l'elenco dei pin utilizzati e la relativa funzione assegnata.

Board Master				
Component	Pinout			
	ECHO		TRIGGER	
Ultrasonic Sensor Left	TIM2_CH1 → PA15			PA0
Ultrasonic Sensor Center	TIM2_CH2 → PB3			PA1
Ultrasonic Sensor Right	TIM2_CH3 → PB10			PA4
	TX	SESSION	MTALK	STALK
Board Communication	USART3_TX → PC10	PC11	PC9	PC8
PWM				
MOTOR FA	TIM1_CH2 → PA9			
MOTOR FB	TIM1_CH1 → PA8			
MOTOR BA	TIM1_CH4 → PC3			
MOTOR BB	TIM1_CH3 → PC2			
COMMAND PIN				
SELECT	PA6			
RELAY	PA7			
	SDA		SCL	
I2C Bus	PB9		PB8	

Tabella C.2: Pinout Configuration for Board Master

### C.2.3 Schema di collegamento complessivo

La figura seguente riporta lo schema complessivo di collegamento del sistema, realizzato in ambiente Fritzing. Esso rappresenta il cablaggio fisico tra le schede e i dispositivi utilizzati. All'interno dello schema sono incluse anche le PCB progettate nell'ambito del lavoro. La loro presenza riflette l'architettura hardware effettivamente implementata e non ha solo finalità illustrativa.

Le connessioni tra i pin delle schede e le PCB seguono quanto definito negli schematici elettrici delle rispettive HAT. Lo schema complessivo deve quindi essere interpretato come una rappresentazione del cablaggio coerente con il progetto elettronico formale.

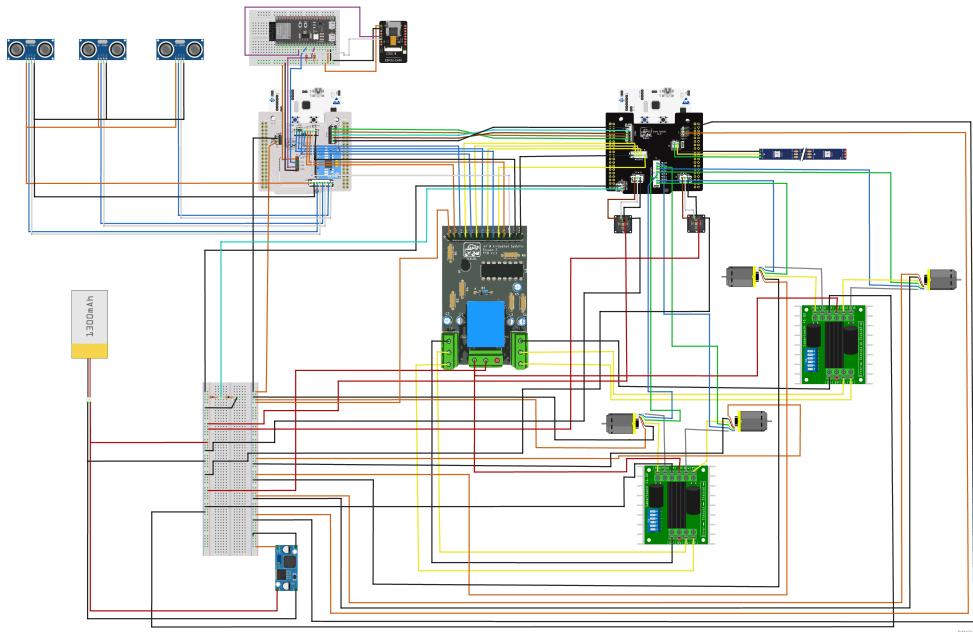


Figura C.3: Schema di collegamento complessivo del sistema

Al fine di rendere esplicita la corrispondenza tra cablaggio fisico e progetto elettronico, sono riportati gli schematici delle PCB HAT progettate.

Tali schematici costituiscono il riferimento formale per la definizione dei collegamenti dei pin e per l'instradamento dei segnali.

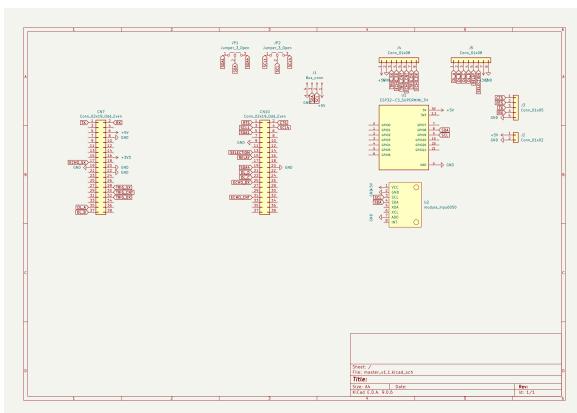


Figura C.4: Schematico elettrico della Board 2 (Master)

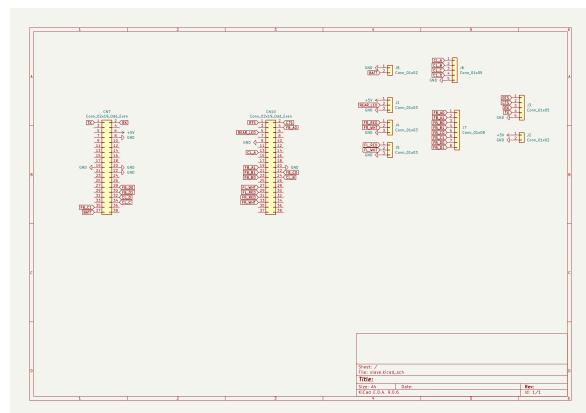


Figura C.5: Schematico elettrico della Board 1 (Slave)