

# Supervisor

Febbraio 2026

# Contents

<b>1</b>	<b>Funzionamento generale di ciascuna board</b>	<b>3</b>
1.1	Interazione tra Task e Meccanismo di Snapshot . . . . .	3
1.2	Interazione tra le due Board . . . . .	5
1.3	Funzionamento task di Board 2 . . . . .	6
1.3.1	Task: acquisizione distanza dagli ostacoli . . . . .	6
	Utilizzo DMA per la lettura dei segnali . . . . .	7
1.3.2	Task: lettura comandi utente . . . . .	9
1.3.3	Task: lettura giroscopio . . . . .	11
1.3.4	Task: supervisore Board 2 . . . . .	12
1.3.5	Task: log dei dati per il debug . . . . .	15
1.4	Funzionamento task di Board 1 . . . . .	16
1.4.1	Task: lettura batteria e temperatura . . . . .	16
1.4.2	Task: supervisore Board 1 . . . . .	18
<b>2</b>	<b>Supervisore Board 1</b>	<b>20</b>
2.1	Panoramica generale . . . . .	20
2.2	Input . . . . .	21
2.3	Output . . . . .	23
2.4	Rilevazione Faults . . . . .	24
2.5	Decidere di far comandare la Board2 . . . . .	28
2.6	Costruzione maschere degradate e critiche . . . . .	30
2.7	Calcolo riferimenti . . . . .	31
<b>3</b>	<b>Supervisore Board 2</b>	<b>33</b>
3.1	Panoramica generale . . . . .	33
3.2	Input . . . . .	34
3.3	Output . . . . .	34
3.4	Rilevamento ostacoli . . . . .	34
3.4.1	Gestione ostacoli con sistema in stato <i>non degradato</i> . . . . .	35
3.4.2	Gestione ostacoli con sistema in stato <i>degradato</i> . . . . .	39

# 1 Funzionamento generale di ciascuna board

## 1.1 Interazione tra Task e Meccanismo di Snapshot

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

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

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

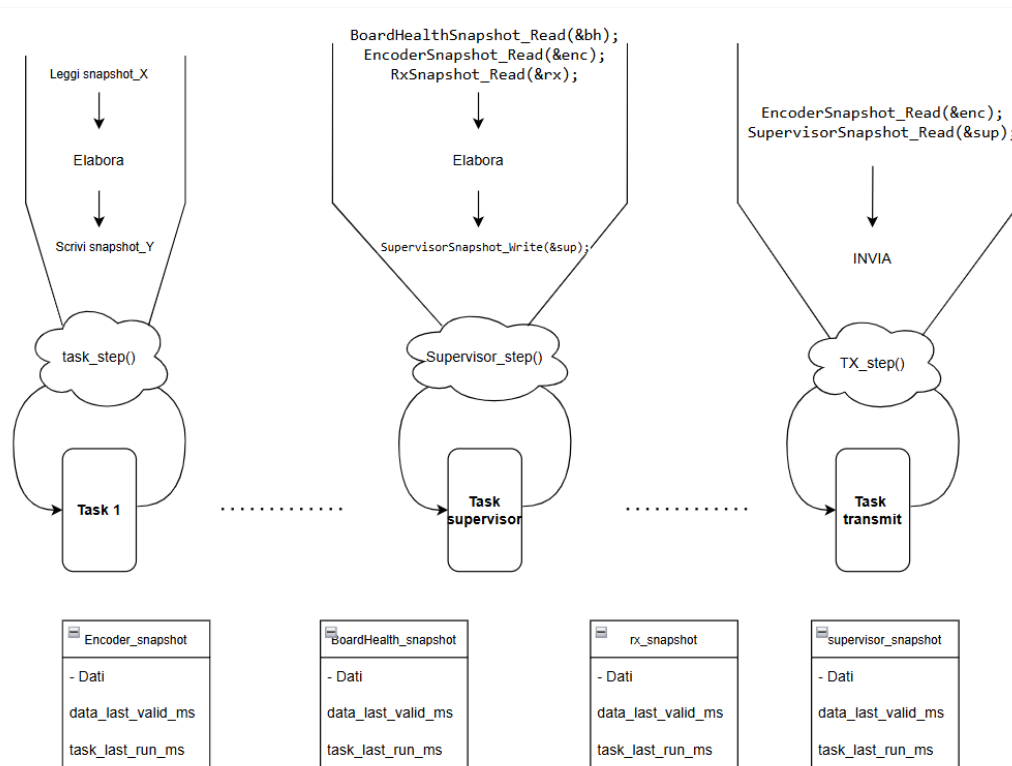


Figure 1: Funzionamento generale del software di Board 1.

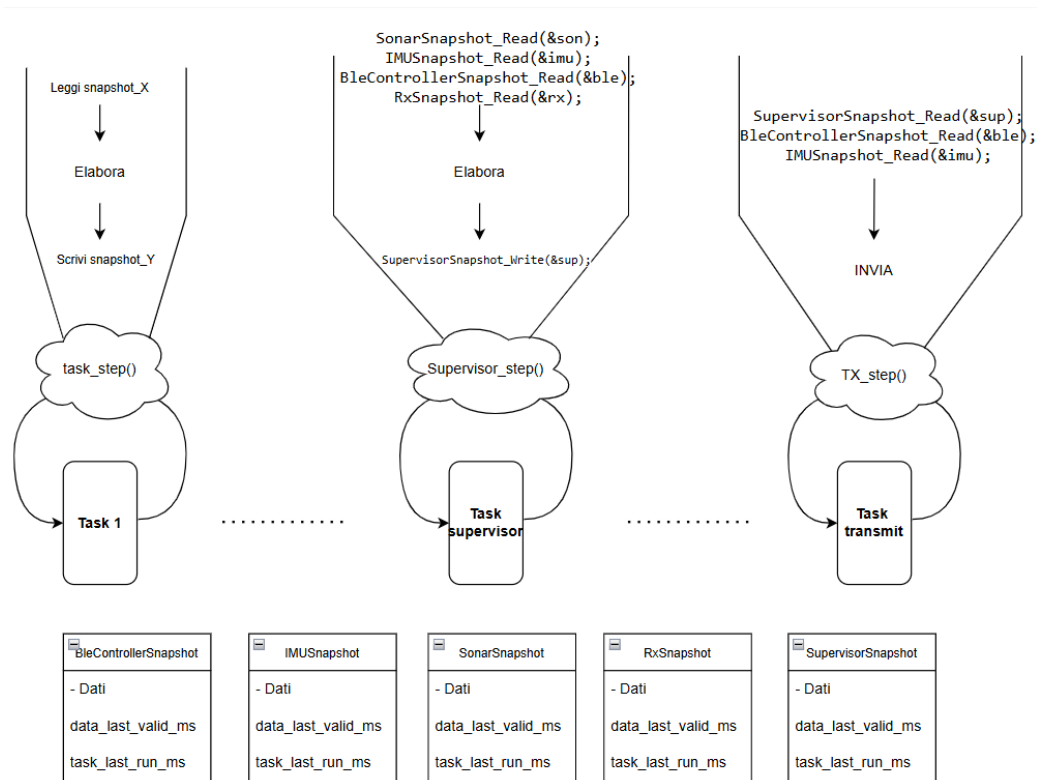


Figure 2: Funzionamento generale del software di Board 2.

## 1.2 Interazione tra le due Board

Le due board comunicano tra loro tramite un collegamento seriale, con presenza di task di trasmissione e ricezione su entrambe le board. Il task di trasmissione su entrambe le board ha periodo di 20 ms, mentre il task di ricezione è di tipo event-driven, attivato dall'arrivo di nuovi dati sulla seriale. In Figura ?? è mostrata l'interazione tra i task di trasmissione della Board 2 e ricezione della Board 1, evidenziando quali sono i dati che vengono trasmessi e ricevuti tra le due board.

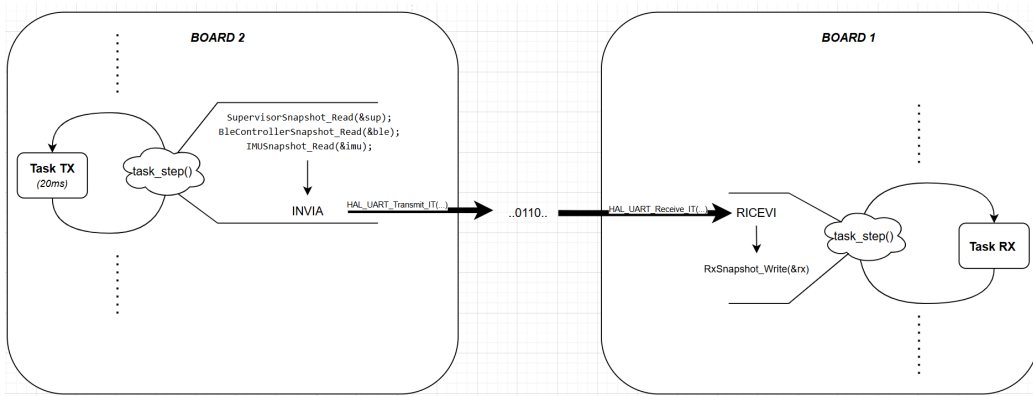


Figure 3: Interazione tra i task di trasmissione della Board 2 e ricezione della Board 1.

Allo stesso modo, nella Figura 4 è mostrata l'interazione tra i task di trasmissione della Board 1 e ricezione della Board 2.

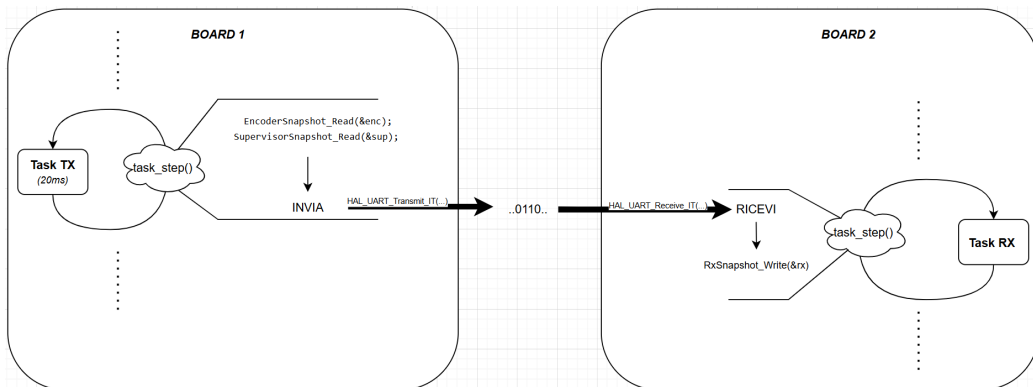


Figure 4: Interazione tra i task di trasmissione della Board 1 e ricezione della Board 2.

## 1.3 Funzionamento task di Board 2

### 1.3.1 Task: acquisizione distanza dagli ostacoli

La Board 2 è equipaggiata con tre sensori ad ultrasuoni **HC-SR04** per il rilevamento di ostacoli. Questi sono disposti a 45° l'uno dall'altro, come mostrato in Figura 5.

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

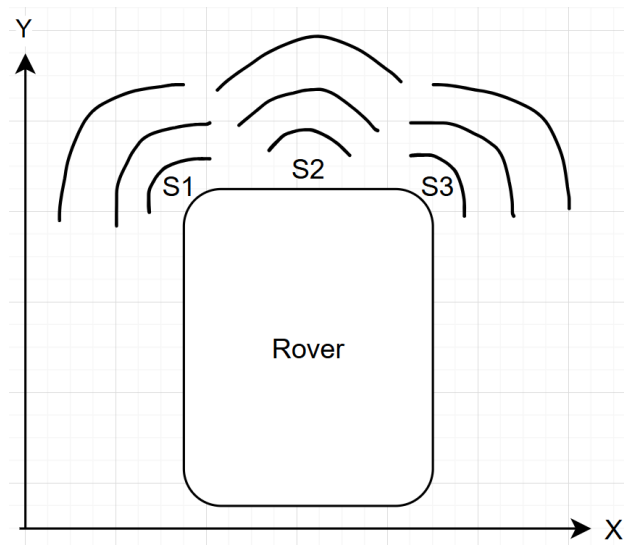


Figure 5: Disposizione dei sensori sulla Board 2.

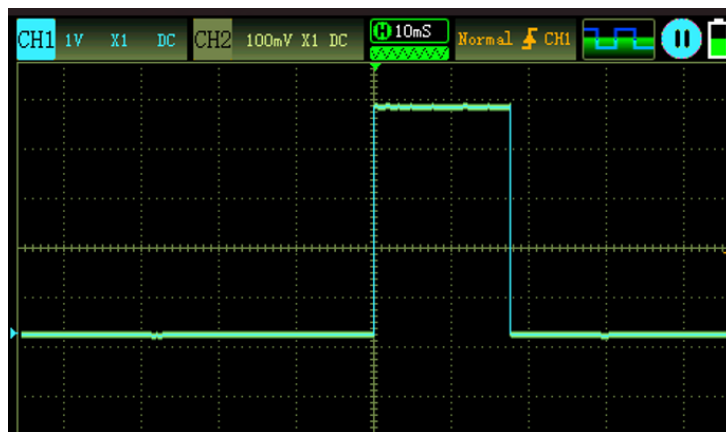


Figure 6: Segnale generato dal sensore HC-SR04 in presenza di un ostacolo a 3 metri di distanza.

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

**Utilizzo DMA per la lettura dei segnali** Per ottimizzare la lettura dei segnali dai sensori ad ultrasuoni, la Board 2 utilizza il Direct Memory Access (DMA). Il DMA consente di trasferire i dati direttamente tra la periferica (i sensori ad ultrasuoni) e la memoria, senza l'intervento della CPU. Il timer

utilizzato è il *Timer1*, con i canali 1, 2 e 3 configurati in modalità *input capture* per catturare i fronti di salita e discesa generati dai tre sensori.

DMA Request	Channel	Direction	Priority
TIM1_CH1	DMA1 Channel 1	Peripheral To Memory	Low
TIM1_CH2	DMA1 Channel 2	Peripheral To Memory	Low
TIM1_CH3	DMA1 Channel 3	Peripheral To Memory	Low

**DMA Request Settings**

Mode: 
 Increment Address: ☐
 Peripheral: ☐
 Memory: ☒

Data Width:

**DMA Request Synchronization Settings**

Enable synchronization: ☐
 Synchronization signal: 
 Signal polarity: 
 Enable event: ☐
 Request number:

Figure 7: Configurazione del DMA per la lettura dei segnali dai sensori.

Ogni canale del DMA è configurato in modalità interrupt, permettendo, alla fine della rilevazione dei due fronti (salita e discesa), di eseguire una *Callback* che imposta dei flag a 1. Questo flag indica che i fronti sono stati rilevati e che la distanza dall'ostacolo può essere calcolata. In totale vengono eseguite solo 3 callback, attivate solo quando uno specifico canale DMA ha terminato la lettura di entrambi i fronti. La Figura 8 mostra un esempio di callback eseguita al termine della rilevazione dei fronti.

```

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

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

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

```

Figure 8: Callback eseguita al termine della rilevazione dei fronti.

Alla fine della lettura, il task aggiorna lo snapshot **SonarsSnapshot** con le distanze rilevate dai tre sensori, come mostrato in Figura 9.

```

typedef struct
{
    uint16_t dist_cm[3];

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

```

Figure 9: Struttura dati dello snapshot SonarsSnapshot.

### 1.3.2 Task: lettura comandi utente

La Board 2 riceve i comandi utente provenienti da un joystick che comunica con un'ESP32 tramite Bluetooth. La Board 2, attraverso un task dedicato, riceve i comandi provenienti dalla *ESP32* con il protocollo **I2Ce** li integra nello snapshot **BleControllerSnapshot**. In Figura 10 è mostrata lo step del task dedicato alla ricezione dei comandi utente, in cui viene eseguita la lettura dei dati provenienti dalla *ESP32* e l'aggiornamento dello snapshot **BleControllerSnapshot**.

```

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

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

    BleI2CStatus_t st = BleController_I2C_ReadFrame(&frame);

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

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

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

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

    BleControllerSnapshot_Write(&snap);
}

```

Figure 10: Step del task dedicato alla ricezione dei comandi utente.

La funzione *BleController\_I2C\_ReadFrame* si occupa di leggere i dati provenienti dalla *ESP32* e di restituirli in una struttura dati, che viene poi utilizzata per aggiornare lo snapshot **BleControllerSnapshot**.

In Figura 11 è mostrata la struttura dati dello snapshot **BleControllerSnapshot**, in cui sono presenti i comandi utente normalizzati ( $x_{norm}$  e  $y_{norm}$ ).

```

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

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

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

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

```

Figure 11: Struttura dati dello snapshot BleControllerSnapshot.

### 1.3.3 Task: lettura giroscopio

La Board 2 è equipaggiata con un sensore IMU **MPU-6050**, che integra un accelerometro e un giroscopio a 3 assi. Poiché il sensore non fornisce direttamente l'orientamento assoluto, l'angolo di yaw (la rotazione del rover attorno al suo asse verticale) viene calcolato nel firmware integrando nel tempo i dati della velocità angolare provenienti dal giroscopio. All'accensione, il sistema esegue una calibrazione per impostare lo zero relativo alla direzione di avvio. La comunicazione con l'IMU avviene tramite protocollo I2C, e i dati elaborati aggiornano ciclicamente lo snapshot IMUSnapshot (Figura 12).

```

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

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

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

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

    float yaw;
} IMUSnapshot_t;

```

Figure 12: Struttura dati dello snapshot IMUSnapshot.

#### **1.3.4 Task: supervisore Board 2**

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

In Figura 13 è mostrato lo step del task del supervisore, in cui vengono evidenziati gli input e gli output.

```

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

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

    uint32_t now = osKernelGetTickCount();

    CommPayloadB1_t payload = rx.payload;

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

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

    SupervisorB2_step();

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

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

```

Figure 13: Step del task del supervisore della Board 2.

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

mando di *CMD\_STOP*. In caso di ostacoli in movimento (range 75 – 150 cm), il supervisore attiva delle procedure di deviazione (*CMD\_GO\_LEFT*, *CMD\_GO\_RIGHT*) o di aggiramento (*CMD\_AVOID*), calcolando i nuovi target di angolo yaw necessari per direzionare il rover verso l'assenza di ostacoli.

Il supervisore esegue un test di coerenza tra la rotazione stimata dalla lettura della velocità dei motori e quella misurata dall'IMU. Questa logica permette di rilevare anomalie meccaniche (come lo slittamento di una ruota o il bloccaggio di un motore) o anomalie dei sensori IMU ed encoder. Se ci sono incoerenze, il supervisore di Board 2 imposta un flag *isMotionConsistent* a 0.

Infine, in condizioni nominali, la Board 2 non invia comandi di attuazione ai motori, lasciando che sia la Board 1 ad attuare. Tuttavia, il controllo attivo dei motori passa alla Board 2 se:

- La Board 1 segnala un guasto interno tramite il proprio payload, autorizzando uno switch di attuazione.
- Viene rilevato un timeout critico nella ricezione dei dati dalla Board 1 ( $> 150$  ms).
- Il supervisore della Board 1 funziona in maniera discontinua.

La descrizione di come il comando possa essere dato da Board 2 in caso di anomalie sarà descritto in seguito.

In Figura 14 è mostrata la *actuation\_step* di Board 2 in funzione della variabile *authorized\_to\_send\_command* in uscita dal supervisore.

```

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

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

        Actuation_Step(v_ref, omega_ref);

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

SupervisorSnapshot_Write(&sup);
}

```

Figure 14: Step di attuazione di Board 2 in funzione della variabile `authorized_to_send_command`.

Data l'assenza degli encoder, l'attuazione di Board 2 dovrà essere in open loop, basandosi esclusivamente sui comandi utente ricevuti e sui dati dei sensori, senza feedback diretto sulla velocità effettiva del rover. La funzione di attuazione trasforma i riferimenti di velocità del supervisore in segnali di potenza per i motori. Il sistema opera in anello aperto (open-loop), convertendo i giri al minuto (RPM) desiderati in tensione elettrica. Infine, i comandi vengono inviati al driver di potenza Sabertooth tramite una scalatura lineare del voltaggio in uscita, con un range di  $[-12V; +12V]$  corrispondente a  $[-MAX\_RPM; +MAX\_RPM]$ .

### 1.3.5 Task: log dei dati per il debug

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

- RxSnapshot
- BleControllerSnapshot
- SonarsSnapshot
- IMUSnapshot

e stampa i dati più rilevanti su console. Questa funzionalità è fondamentale per monitorare lo stato del sistema durante le fasi di sviluppo e test, permettendo di identificare rapidamente eventuali anomalie o comportamenti imprevisti. Lo step del task è mostrato in Figura 15, in cui vengono evidenziati i dati acquisiti e stampati su console.

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

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

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

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

    printf("%s", log_buf);
}
```

Figure 15: Step del task dedicato alla stampa dei dati per scopi di debug.

## 1.4 Funzionamento task di Board 1

### 1.4.1 Task: lettura batteria e temperatura

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

In Figura 16 è mostrato lo step del task dedicato alla lettura della batteria e temperatura, in cui vengono evidenziati i dati acquisiti e aggiornati nello snapshot **BoardHealthSnapshot**.

```

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

    BoardHealthStatus_t temp_st;
    BoardHealthStatus_t batt_st;

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

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

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

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

    BoardHealthSnapshot_Write(&snap);
}

```

Figure 16: Step del task dedicato alla lettura della batteria e temperatura.

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

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

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

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

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

#### 1.4.2 Task: supervisore Board 1

Il supervisore della Board 1 è implementato anch'esso tramite una macchina a stati sviluppata in Simulink Stateflow. Questo task ha il compito di supervisionare l'integrità dell'hardware e di orchestrare le logiche di movimento avanzate.

In Figura 17 è illustrato lo step del task del supervisore della Board 1, con il dettaglio delle interfacce di input e output.

```

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

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

    uint32_t now = osKernelGetTickCount();

    CommPayloadB2_t payload = rx.payload;

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

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

    SupervisorB1_step();

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

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

```

Figure 17: Step del task del supervisore della Board 1.

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

La logica di controllo è strutturata su tre pilastri fondamentali:

- **Diagnostica e Fault Masking:** Il sistema aggrega i guasti rilevati in maschere di errore (*Critical* e *Degraded*). Queste maschere permettono

al supervisore di inibire determinate manovre o di limitare la potenza del sistema in caso di anomalie termiche o di tensione, garantendo la protezione della componentistica meccanica ed elettronica.

- **Gestione delle Manovre Complesse:** A differenza della Board 2, focalizzata sull'evitamento ostacoli, la Board 1 gestisce comandi di alto livello come la *Rotazione a 180 gradi* assistita e la procedura di *Emergency Stop*, monitorando l'arresto effettivo delle ruote tramite i sensori di velocità.
- **Arbitraggio dell'Attuazione:** Il supervisore valuta costantemente l'affidabilità della comunicazione inter-board. Qualora rilevi un'instabilità critica nel proprio sottosistema di controllo ma una comunicazione valida con l'altra scheda, è in grado di cedere i privilegi di attuazione alla Board 2, realizzando una strategia di ridonanza funzionale che aumenta la resilienza del rover.

## 2 Supervisore Board 1

### 2.1 Panoramica generale

Il supervisore della Board 1 è implementato come un modulo Simulink denominato **SupervisorB1**, il cui schema a blocchi è illustrato in Figura 18.

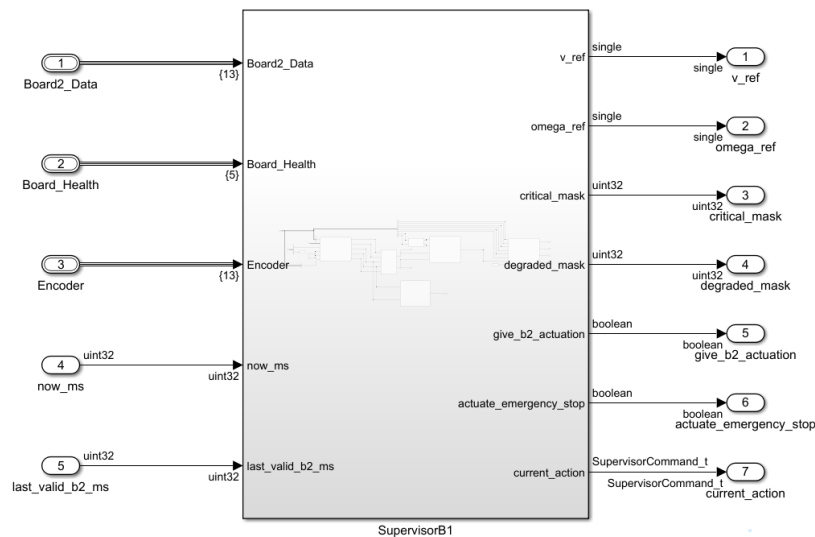


Figure 18: Schema a blocchi del modulo SupervisorB1.

Il suo compito è quello di decidere il riferimento di velocità ( $v_{ref}$ ) e di direzione ( $\omega_{ref}$ ) del rover, in funzione dell'elaborazione dei dati di input. In particolare, esso è composto da quattro parti principali:

- **Rilevazione Faults:** si occupa di rilevare anomalie nella ricezione dei dati da Board2, dagli encoder delle ruote, dai sensori di temperatura e batteria.
- **Aggregazione Fault:** aggrega le anomalie rilevate nella parte di *rilevazione faults* e le codifica in due maschere di errore (critica e degradata).
- **Decidere di far comandare la Board2:** decide se autorizzare o meno la Board2 a muovere il rover in base alle condizioni di fault rilevate.
- **Calcolo Riferimenti:** se Board1 attua, calcola i riferimenti di velocità lineare e angolare del rover in base ai comandi ricevuti dalla Board2 e alle condizioni di fault rilevate.

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

## 2.2 Input

I segnali in input che riceve sono:

- **Board2\_Data:** è l'ultimo snapshot di ricezione (*RxSnapshot*) aggiornato dal task di ricezione di Board1. Alcuni dei segnali utilizzati dal supervisore di Board 1 sono:
  - *command*: rappresenta il comando in uscita dal supervisore della Board2, che può assumere i seguenti valori:

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

Figure 19: Comandi in uscita dal supervisore della Board2.

- *x\_norm* & *y\_norm*: rappresenta il comando utente proveniente dal joystick. Così se il comando *command* proveniente da Board 2 è *CMD\_NORMAL* e la il supervisore di Board 1 non rileva anomalie, i riferimenti di velocità lineare e angolare del rover vengono rispettivamente utilizzando *x\_norm* e *y\_norm* ( $speed\_ref = y\_norm \cdot MAX\_SPEED$  e  $steering\_cmd = x\_norm \cdot MAX\_TURN$ ).
- *yaw*: rappresenta l'angolo di orientamento del rover, calcolato a partire dai dati provenienti dalla Board2.
- *isMotionConsistent*: Verifica la coerenza tra la rotazione misurata dalla IMU e quella stimata dai motori, per rilevare guasti meccanici o slittamenti .
- *critical\_mask* & *degraded\_mask*: sono due maschere di errore a 8 bit calcolate da Board 2, in cui ogni bit indica la presenza di un'anomalia *critica* (da cui *critical\_mask*) o *degradata* (da cui *degraded\_mask*) specifica.
- **Board\_Health**: è l'ultimo snapshot di salute del sistema (*BoardHealthSnapshot*), in cui sono presenti i valori di temperatura e batteria. La struttura dati è la seguente:

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

    uint32_t task_last_run_ms; /* ultima esecuzione del task */

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

Figure 20: Struttura dati Board\_Health.

- **Encoder**: è l'ultimo snapshot delle velocità delle ruote (*wheel\_speed\_rpm*) e di eventuali anomalie riscontrate in uno dei motori (*has\_no\_feedback*). La struttura dati è la seguente:

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

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

Figure 21: Struttura dati Encoder.

- **now\_ms**: Rappresenta il tempo corrente in millisecondi.
- **last\_valid\_b2\_ms**: Rappresenta il tempo in millisecondi dell'ultimo dato valido ricevuto dalla Board2.

## 2.3 Output

I segnali in output che fornisce sono:

- **v\_ref**: rappresenta il riferimento di velocità lineare del rover, in m/s.
- **omega\_ref**: rappresenta il riferimento di velocità angolare del rover, in rad/s.
- **critical\_mask & degraded\_mask**: sono due maschere di errore a 8 bit, in cui ogni bit indica la presenza di un'anomalia *critica* (da cui *critical\_mask*) o *degradata* (da cui *degraded\_mask*) specifica, come descritto nella Tabella 1.

Bit	ID Segnale	Descrizione dell'Anomalia
0	TEMP_CRI/DEG	Temperatura fuori range o monitoraggio temperatura assente
1	BATT_CRI/DEG	Tensione batteria fuori range o monitoraggio tensione assente
2	COMM_CRI/DEG	Comunicazione lenta o assente
3-6	WHEEL_CRI/DEG	Guasti attuatori (FL, FR, RL, RR)
7	SUP_CRI/DEG	Supervisore B2 lento o assente

Table 1: Mappatura della maschera di errore a 8 bit

- **give\_b2\_actuation**: rappresenta un segnale booleano che indica se la Board2 deve essere autorizzata a muovere il rover.
- **actuate\_emergency\_stop**: rappresenta un segnale booleano che indica se il rover deve essere fermato immediatamente per evitare collisioni.
- **current\_action**: rappresenta un segnale che indica l'azione attualmente intrapresa dal supervisore.

## 2.4 Rilevazione Faults

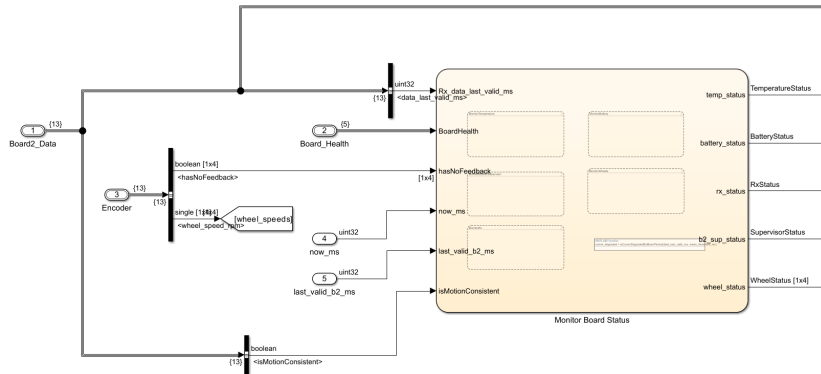


Figure 22: Chart di gestione dei faults.

L'output di questo sottosistema è rappresentato da 5 segnali, ognuno dei quali indica la presenza o meno di un'anomalia *critica* o *degradata* specifica proveniente dalla lettura dei segnali degli encoder, batteria, temperatura e comunicazione con la Board2. Di seguito si descrivono i valori che questi segnali possono assumere e le condizioni che portano a tali valori.

Table 2: Stati degradato e critico per Board 1

OUTPUT	VALORE	SIGNIFICATO
temp_status	TEMP_HEALTH_DEGRADED	Quando la temperatura è nel range $] - 15; -5] \cup [55; +60[$
	TEMP_HEALTH_CRITICAL	<ul style="list-style-type: none"> <li>Quando la temperatura è per almeno 4s nel range  <math>] - \infty; -15] \cup [60; +\infty[</math></li> <li>Nel caso in cui l'intervallo di tempo dall'ultimo aggiornamento della temperatura è di 0.5s, si ipotizza che la temperatura aumenti di 1 °C/s. Se questa sale raggiungendo un valore superiore a 65 °C/s si ha questo valore.</li> </ul>

OUTPUT	VALORE	SIGNIFICATO
	TEMP_HEALTH_OK	Quando la temperatura è nel range $] - 5; 55[$
<b>batt_status</b>	BATT_HEALTH_DEGRADED	Quando la percentuale di batteria è minore del 23%
	BATT_HEALTH_CRITICAL	<ul style="list-style-type: none"> <li>• Quando la temperatura è per almeno 5 s nel range <math>[0\%; 15\%]</math></li> <li>• Nel caso in cui l'intervallo di tempo dall'ultimo aggiornamento della percentuale batteria è di 0.5 s, si ipotizza che la percentuale diminuisca di 0.42%/s. Se questa diminuisce raggiungendo un valore minore a 15% si ha questo valore.</li> </ul>
	BATT_HEALTH_OK	Quando la percentuale è $> 25\%$
<b>wheel_status(i)</b>	WHEEL_DEGRADED_ENCODER	Viene attivato dalla funzione <i>has_no_feedback(i)</i> quando si rileva un'assenza di impulsi dall'encoder nonostante il motore sia alimentato. Questa condizione indica un guasto al sensore o un blocco meccanico parziale. Questa condizione permette l'attivazione di una logica di recupero ( <i>fallback</i> ) che ricostruisce la velocità della ruota guasta basandosi sulla lettura dei sensori integri degli altri motori.

OUTPUT	VALORE	SIGNIFICATO
	WHEEL_CRITICAL_MOTOR	Questo stato viene attivato quando alla perdita del feedback dell'encoder si somma un'incoerenza tra il dato del sensore IMU e la rotazione stimata tramite l'odometria delle ruote. Tale discrepanza, rilevata dal controllo globale <i>isMotionConsistent</i> , indica che il sistema di fallback non è più in grado di garantire una stima affidabile del moto.
	WHEEL_OK	Rappresenta la condizione di pieno funzionamento del sottosistema. In questo stato, il sistema opera in assenza di anomalie sia a livello locale, dove la velocità calcolata dagli encoder risulta coerente con il comando impartito (verificato tramite <i>has_no_feedback</i> ), , sia a livello globale, dove i dati del sensore IMU confermano la rotazione stimata in base agli RPM delle ruote (verificato tramite <i>isMotionConsistent</i> )

OUTPUT	VALORE	SIGNIFICATO
b2_sup_status	SUP_DEGRADED	Questo valore indica una discontinuità operativa della Board 2. Il supervisore della Board 1 monitora l'heartbeat (una variabile) del supervisore remoto, il quale incrementa il valore ogni volta che viene eseguito. Se la media degli ultimi 10 intervalli di aggiornamento superi i 40 ms, il sistema segnala uno stato di degrado del supervisore di Board 2.
	SUP_CRITICAL	Se l'intervallo di tempo dall'ultimo aggiornamento dell'heartbeat del supervisore di Board2 supera i 120 ms, imposta questo valore.
	SUP_OK	Quando non ci sono ne condizioni critiche ne degradate imposta questo valore.

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

## 2.5 Decidere di far comandare la Board2

Il sottosistema per decidere se autorizzare o meno la Board2 a muovere il rover è mostrato in Figura 23.

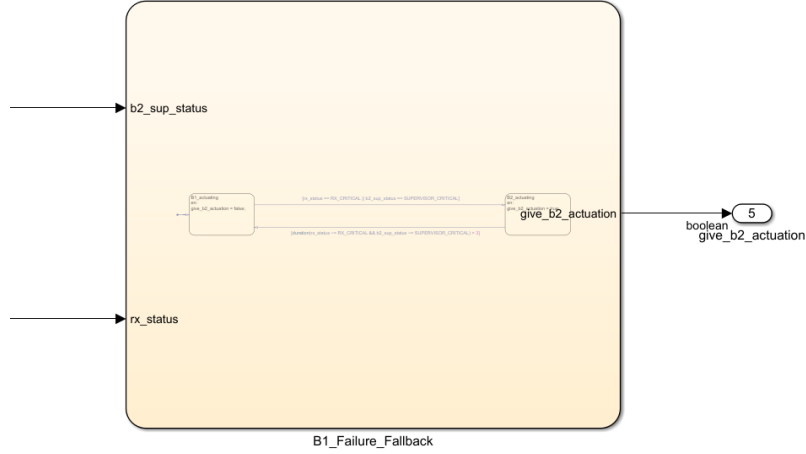
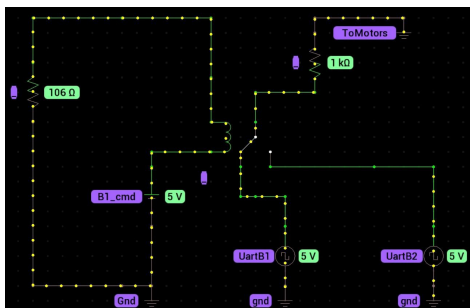


Figure 23: Chart per decidere se autorizzare o meno la Board2 a muovere il rover.

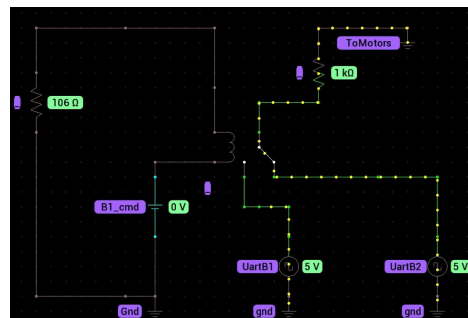
L'autorizzazione a Board 2 per il controllo del rover è regolata dalla variabile *give\_b2\_actuation* secondo la seguente logica:

- **give\_b2\_actuation = 1:** Il relè è aperto e il movimento viene gestito da Board 2. Questa condizione si ha quando sono riscontrate anomalie *critiche* in ricezione (*texttrx\_status* = *RX\_CRITICAL*), *supervisore* (*textitb2\_sup\_status* = *SUP\_CRITICAL*).
- **give\_b2\_actuation = 0:** Il relè è chiuso e il movimento viene gestito da Board 1. Questa condizione si ha quando non sono riscontrate anomalie *critiche* in ricezione (*texttrx\_status* = *RX\_OK*), *supervisore* (*textitb2\_sup\_status* = *SUP\_OK*).

Tale meccanismo agisce come un dispositivo di sicurezza hardware basato sullo stato del supervisore, come mostrato nelle Figure 24b e 24a.



(a) Il comando di Board1 chiude il rele impedendo a Board2 di muovere il rover.



(b) Board1 permette alla Board2 di muovere il rover.

## 2.6 Costruzione maschere degradate e critiche



Figure 25: Chart per la costruzione delle maschere di errore critiche e degradate.

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

Il bit di ogni maschera rappresenta un'anomalia specifica, come descritto nella Tabella 1.

Bit	Componente	critical_mask	degraded_mask
0	Temperatura	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>
1	Batteria	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>
2	Ricevitore (RX)	1 se <b>RX_CRITICAL</b>	1 se <b>RX_DEGRADED</b>
3	Ruota FL	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>
4	Ruota FR	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>
5	Ruota RL	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>
6	Ruota RR	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>
7	B2 Supervisor	1 se <b>TEMP_HEALT_CRITIC</b>	1 se <b>TEMP_HEALT_DEG</b>

Table 3: Mappatura dei Bit nelle Fault Masks

## 2.7 Calcolo riferimenti

Il sottomodello per il calcolo dei riferimenti di velocità lineare e angolare del rover è mostrato in Figura 26.

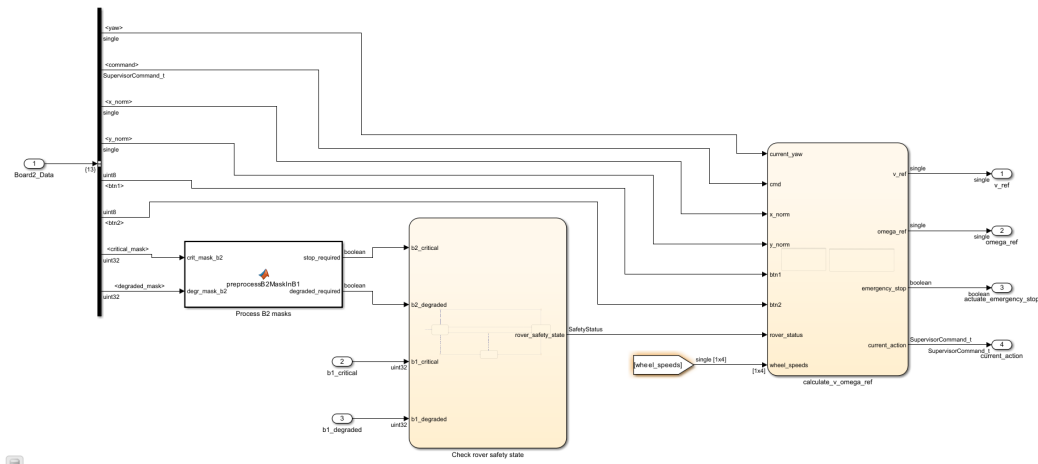


Figure 26: Chart per il calcolo dei riferimenti di velocità lineare e angolare del rover.

In particolare i riferimenti di velocità lineare e angolare del rover vengono calcolati in funzione dei comandi ricevuti dalla Board2 e delle condizioni di fault rilevate. In assenza di anomalie, i riferimenti vengono calcolati direttamente a partire dai comandi ricevuti dalla Board2, in particolare, il riferimento di velocità lineare è calcolato a partire dal comando  $y_{norm}$  e il riferimento di velocità angolare è calcolato a partire dal comando  $x_{norm}$ . Nel caso in cui siano rilevate anomalie, i riferimenti vengono calcolati in modo da ridurre la velocità del rover o addirittura fermarlo, a seconda della

gravità dell'anomalia rilevata. I riferimenti vengono scalati come mostrato in Figura 27.

```
function [V_MAX, OMEGA_MAX, V_MAX_MANEUVER, OMEGA_GO_LEFT, OMEGA_GO_RIGHT] = ...
    updateSafetyLimits(rover_state)

% ---- valori nominali ----
V_MAX      = 1.0;
OMEGA_MAX  = 1.0;
V_MAX_MANEUVER = 0.8;
OMEGA_GO_LEFT  = 0.4;
OMEGA_GO_RIGHT = -0.4;

% ---- scaling in base allo stato ----
if rover_state == SafetyStatus.SAFETY_DEGRADED

    V_MAX      = 0.5 * V_MAX;
    OMEGA_MAX  = 0.5 * OMEGA_MAX;
    V_MAX_MANEUVER = 0.5 * V_MAX_MANEUVER;
    OMEGA_GO_LEFT  = 0.5 * OMEGA_GO_LEFT;
    OMEGA_GO_RIGHT = 0.5 * OMEGA_GO_RIGHT;

elseif rover_state == SafetyStatus.SAFETY_CRITICAL

    V_MAX      = 0;
    OMEGA_MAX  = 0;
    V_MAX_MANEUVER = 0;
    OMEGA_GO_LEFT  = 0;
    OMEGA_GO_RIGHT = 0;

end

end
```

Figure 27: Scaling dei riferimenti in base alla gravità dell'anomalia rilevata.

Nella figura 27 è mostrato come i riferimenti di velocità lineare e angolare vengono scalati in base alla gravità dell'anomalia rilevata, in particolare utilizzando la variabile *safety\_status* che viene calcolata a partire dalle maschere di errore critiche e degradate di entrambe le Board, come mostrato in figura 28.

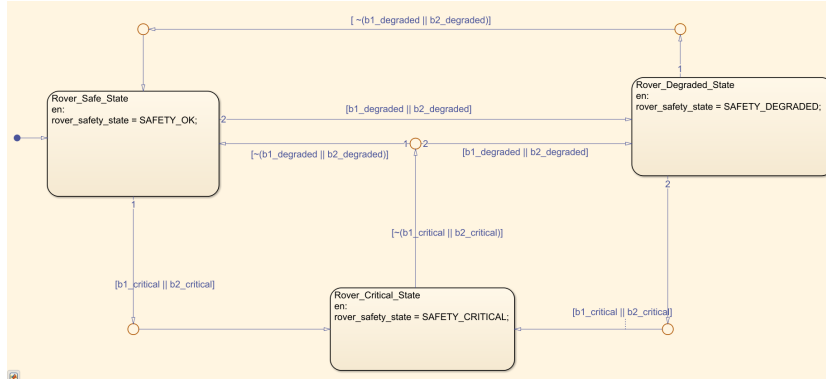


Figure 28: Calcolo della variabile `safety_status` a partire dalle maschere di errore critiche e degradate di entrambe le Board.

## 3 Supervisore Board 2

### 3.1 Panoramica generale

Il supervisore della Board 2 è implementato come un modulo Simulink denominato **SupervisorB2**, il cui schema a blocchi è illustrato in Figura 29.

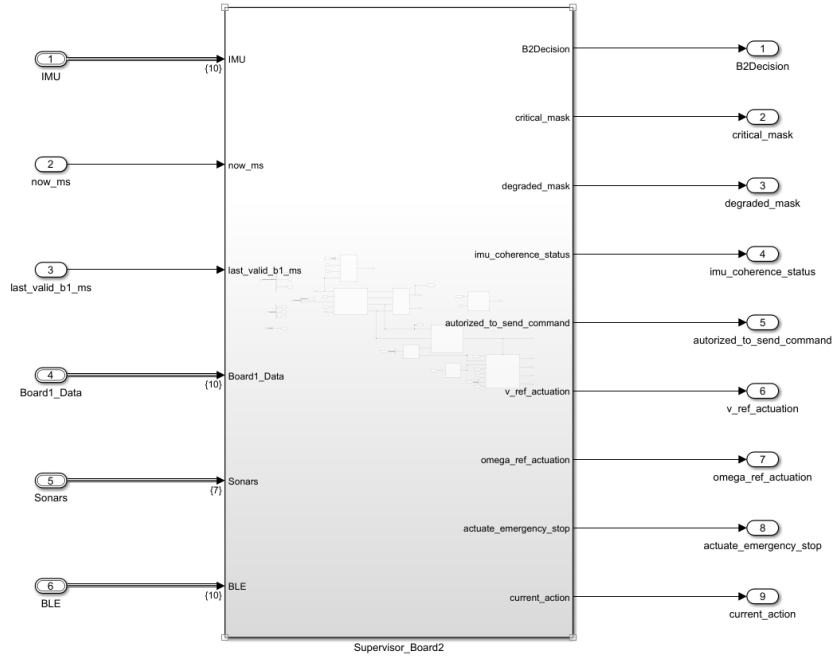


Figure 29: Schema a blocchi del modulo SupervisorB2.

Il suo compito è quello di decidere il riferimento di velocità ( $v_{ref}$ ) e di direzione ( $\omega_{ref}$ ) del rover, in funzione dell'elaborazione dei dati di input. In particolare, esso è composto da 3 parti principali:

- **Gestione Faults:** si occupa di rilevare e gestire eventuali anomalie nei dati ricevuti dalla Board1, dagli encoder delle ruote, dai sensori di temperatura e batteria.
- **Decidere di far comandare la Board2:** decide se autorizzare o meno la Board2 a muovere il rover, in base alle condizioni di fault rilevate.
- **Aggregazione Fault:** aggrega le anomalie rilevate nella parte di gestione faults e le codifica in due maschere di errore (critica e degradata).
- **Calcolo Riferimenti:** calcola i riferimenti di velocità lineare e angolare del rover in base ai comandi ricevuti dalla Board2 e alle condizioni di fault rilevate.

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

### 3.2 Input

I segnali in input che riceve sono:

### 3.3 Output

I segnali in output che fornisce sono:

### 3.4 Rilevamento ostacoli

Come da specifiche, il comportamento del rover in presenza di ostacoli, deve essere regolato sulla base delle condizioni in cui può trovarsi:

#### 1. *Stato non degradato*

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

## 2. Stato degradato

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

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

### 3.4.1 Gestione ostacoli con sistema in stato *non degradato*

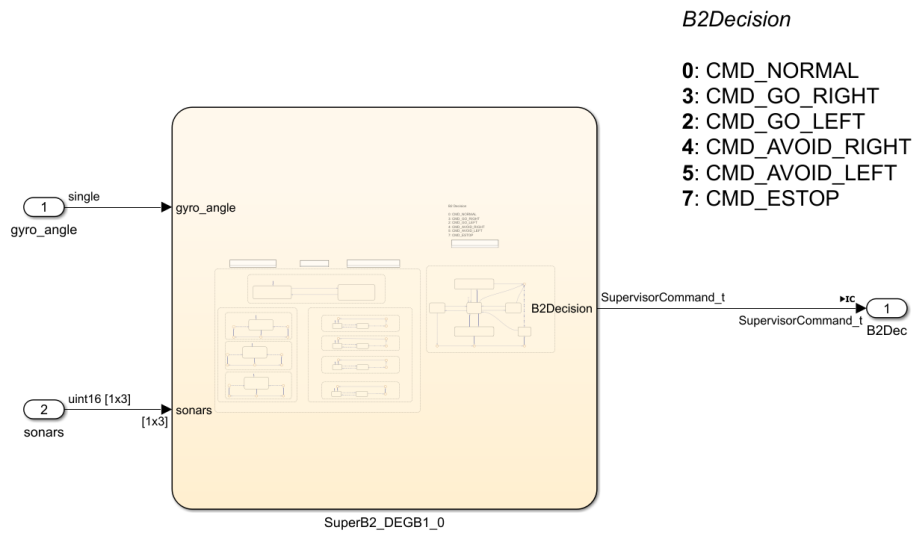


Figure 30: Logica di gestione degli ostacoli in stato non degradato.

Il chart per la gestione degli ostacoli in stato non degradato è mostrato in Figura 31.



rover non ruota di  $45^\circ$  rispetto alla direzione iniziale, verso la direzione del sonar che per primo ha rilevato l'ostacolo.

2. **Sonars**: All'interno di questo stato parallelo sono presenti altri 3 stati paralleli.

(a) **distance\_70**: Rappresenta la condizione in cui uno dei sonar rileva un ostacolo a una distanza  $\leq 70$  cm.

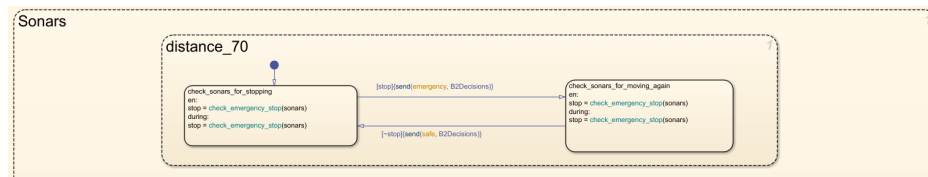


Figure 33: Stato parallelo per la gestione di un ostacolo a distanza  $\leq 70$  cm.

In questo stato quando uno dei sonar rileva un ostacolo a una distanza inferiore o uguale a 70 cm, viene attivata una transizione che porta allo stato di arresto immediato del rover. In particolare, quando un sonar rileva la presenza di un ostacolo a distanza  $\leq 70$  cm, viene inviato un segnale **Emergency** allo stato parallelo *B2Decisione* per fermare il rover. B2Decisione utilizza questo segnale per portarsi nello stato in cui l'output del chart prevede lo stop.

(b) **distance\_gt70** —- **timers**: Questi due stati insieme permettono il rilevamento di un ostacolo in movimento tra le coppie di sonar

- *S1-S2* (tra sonar di sinistra e sonar centrale)
- *S2-S1* (tra sonar centrale e sonar di sinistra)
- *S2-S3* (tra sonar centrale e sonar di destra)
- *S3-S2* (tra sonar di destra e sonar centrale)

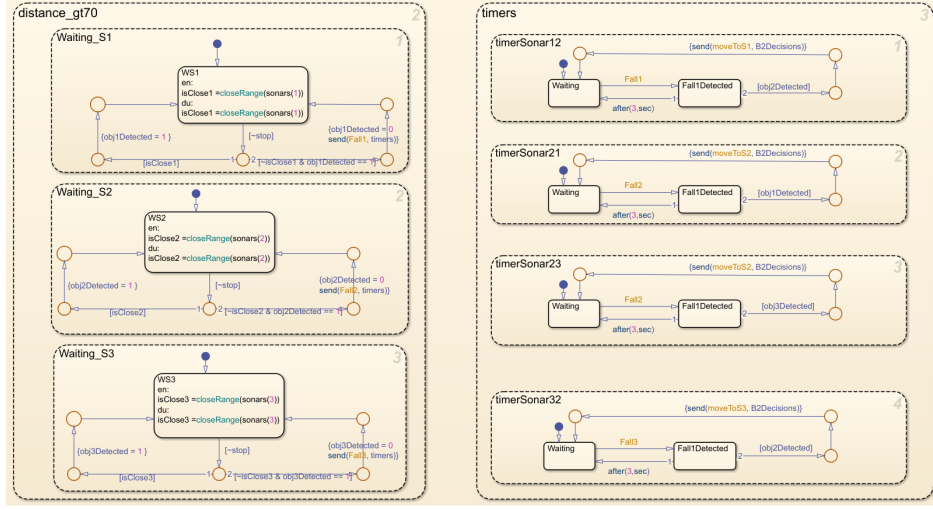


Figure 34: Stato parallelo per la gestione di un ostacolo in movimento a distanza  $> 70$  cm.

Nello stato *distance\_gt70* sono presenti 3 stati paralleli, *Waiting\_S1*, *Waiting\_S2*, *Waiting\_S3*, uno per ogni sonar.

Di seguito si analizza la dinamica di rilevamento di un ostacolo che si sposta dal sonar *S1* verso il sonar *S2*. Tale logica è da considerarsi valida per ogni coppia di sensori precedentemente elencata. Si assume, come condizione necessaria, l'assenza di ostacoli a una distanza inferiore a 70 cm; in caso contrario, il sistema non procederebbe al rilevamento di oggetti in movimento.

- i. **Attivazione (*S1*):** Quando il sonar *S1* rileva un oggetto entro il range 100–300 cm, la variabile *obj1Detected* viene impostata a 1 (**fronte di salita**).
- ii. **Transizione e Timing:** Nel momento in cui l'oggetto esce dal campo d'azione di *S1*, la variabile *obj1Detected* torna a 0 (**fronte di discesa**). Contestualmente, lo stato *timerSonar12* del modulo *timers* avvia un conteggio di 3 secondi.
- iii. **Verifica (*S2*):** Se il sonar *S2* rileva l'ostacolo (sempre tra 100 e 300 cm) entro la finestra temporale dei 3 secondi, viene inviato il segnale *moveToS1* allo stato parallelo *B2Decision*. Qualora il timer scada senza alcun rilevamento da parte di *S2*, non viene trasmesso alcun segnale.

### 3.4.2 Gestione ostacoli con sistema in stato *degradato*

Il chart per la gestione degli ostacoli in stato degradato è mostrato in Figura 35.

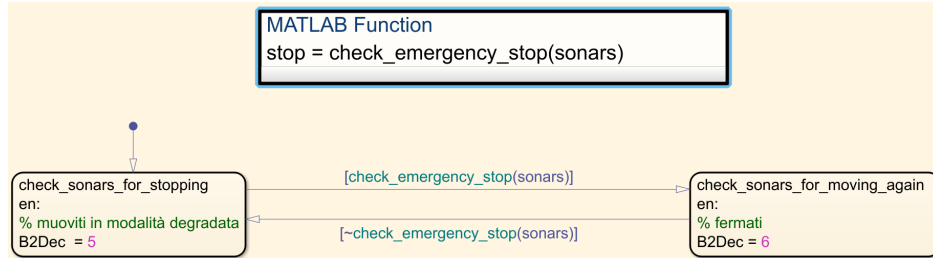


Figure 35: Chart di gestione ostacoli in stato degradato.

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