

# Supervisori

Febbraio 2026

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Acquisizione segnali Board 2</b>                                   | <b>3</b>  |
| 1.1      | HC-SR04 - Sensore ad ultrasuoni . . . . .                             | 3         |
| <b>2</b> | <b>Funzionamento generale di ciascuna board</b>                       | <b>5</b>  |
| 2.1      | Interazione tra Task e Meccanismo di Snapshot . . . . .               | 5         |
| <b>3</b> | <b>Supervisore Board 1</b>  | <b>7</b>  |
| 3.1      | Panoramica generale . . . . .   | 7         |
| 3.2      | Input . . . . .   | 8         |
| 3.3      | Output . . . . .  | 10        |
| 3.4      | Rilevazione Faults . . . . .  | 11        |
| 3.5      | Decidere di far comandare la Board2 . . . . .                         | 15        |
| 3.6      | Costruzione maschere degradate e critiche . . . . .                   | 17        |
| 3.7      | Calcolo riferimenti . . . . .   | 18        |
| <b>4</b> | <b>Supervisore Board 2</b>  | <b>20</b> |
| 4.1      | Panoramica generale . . . . .   | 20        |
| 4.2      | Input . . . . .   | 21        |
| 4.3      | Output . . . . .  | 21        |
| 4.4      | Rilevamento ostacoli . . . . .  | 21        |
| 4.4.1    | Gestione ostacoli con sistema in stato <i>non degradato</i> . . . . . | 22        |
| 4.4.2    | Gestione ostacoli con sistema in stato <i>degradato</i> . . . . .     | 26        |

# 1 Acquisizione segnali Board 2

## 1.1 HC-SR04 - Sensore ad ultrasuoni

La Board 2 è equipaggiata con tre sensori ad ultrasuoni **HC-SR04** per il rilevamento di ostacoli. Questi sono disposti a  $45^\circ$  l'uno dall'altro, come mostrato in Figura 1.

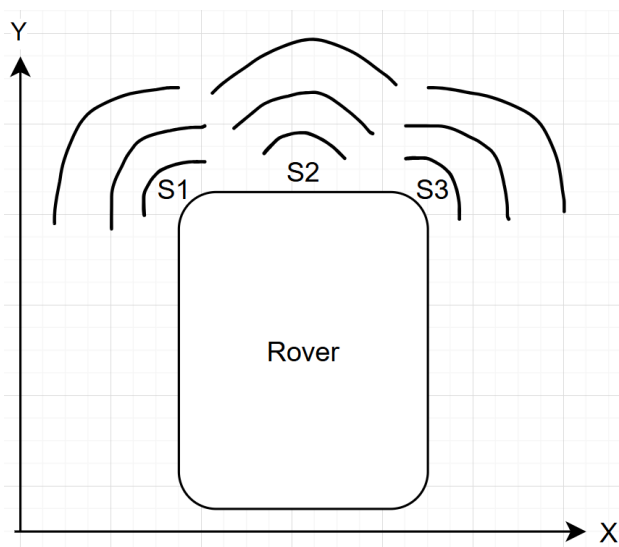


Figure 1: Disposizione dei sensori sulla Board 2.

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

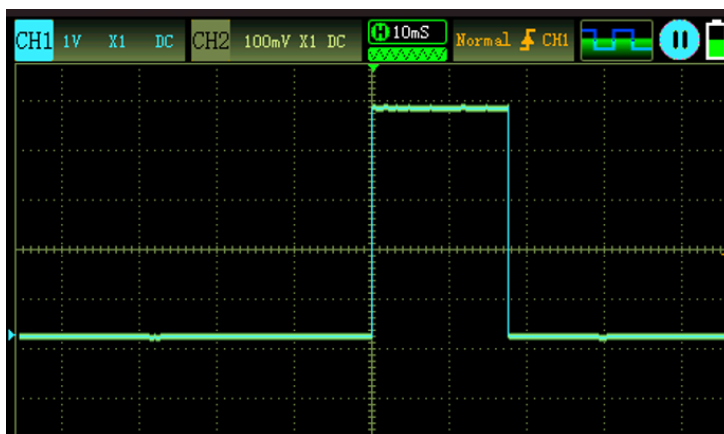


Figure 2: Segnale generato dal sensore HC-SR04 in presenza di un ostacolo a 3 metri di distanza.

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      |

Buttons: Add, Delete

**DMA Request Settings**

Mode: Circular

Increment Address: ☐ Peripheral ☒ Memory

Data Width: Half Word

**DMA Request Synchronization Settings**

Enable synchronization: ☐

Synchronization signal:

Signal polarity:

Enable event: ☐

Request number:

Figure 3: 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 4 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 4: Callback eseguita al termine della rilevazione dei fronti.

## 2 Funzionamento generale di ciascuna board

### 2.1 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 5 che illustra il funzionamento dei task *SupervisorB1* e *Transmit*. 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 e scrive i propri snapshot, come illustrato nella Figura 6.

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 responsabile dell'aggiornamento è stato effettivamente eseguito.

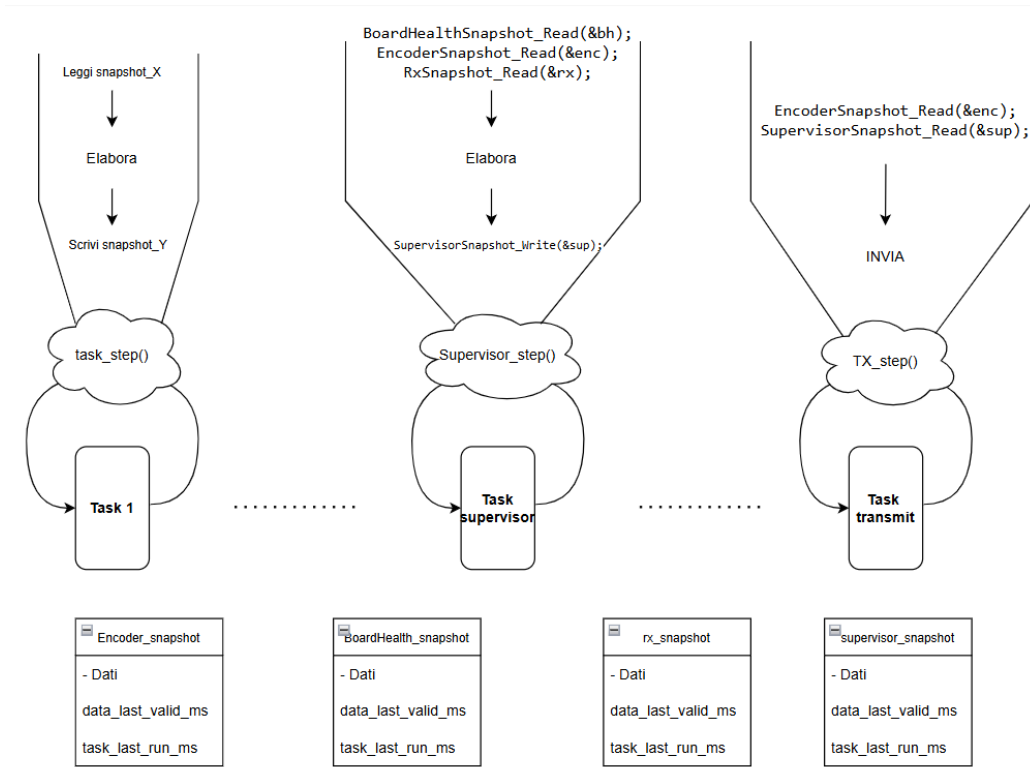


Figure 5: Funzionamento generale del software di Board 1.

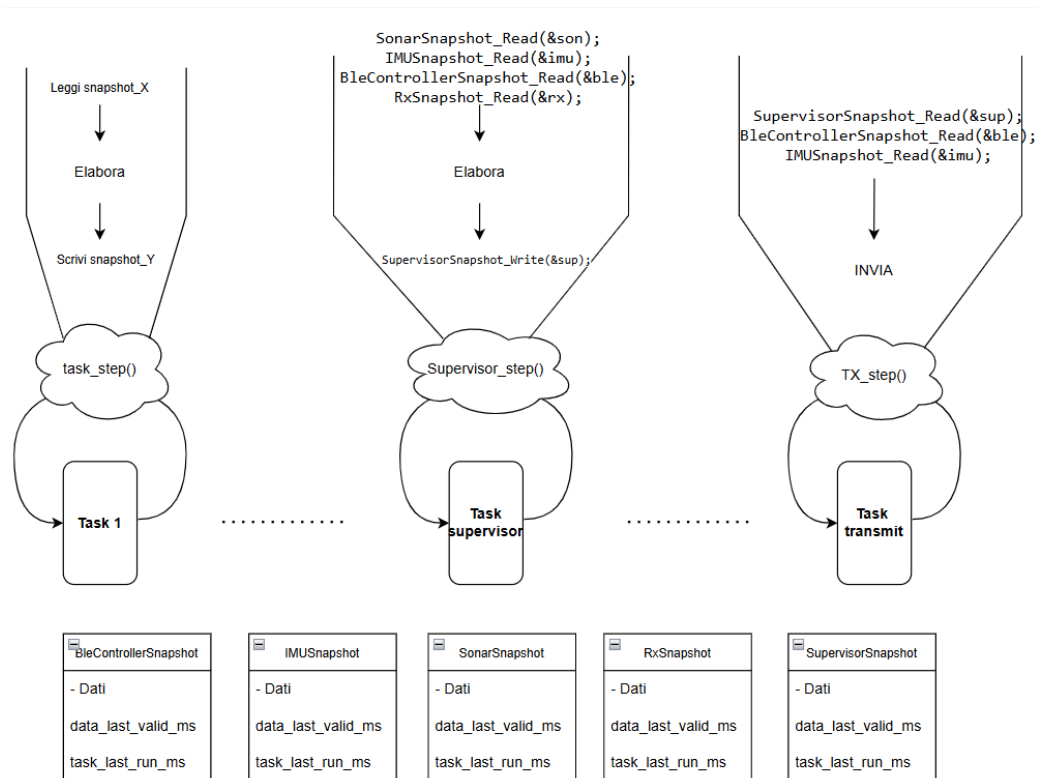


Figure 6: Funzionamento generale del software di Board 2.

## 3 Supervisore Board 1

### 3.1 Panoramica generale

Il supervisore della Board 1 è implementato come un modulo Simulink denominato **SupervisorB1**, il cui schema a blocchi è illustrato in Figura 7.

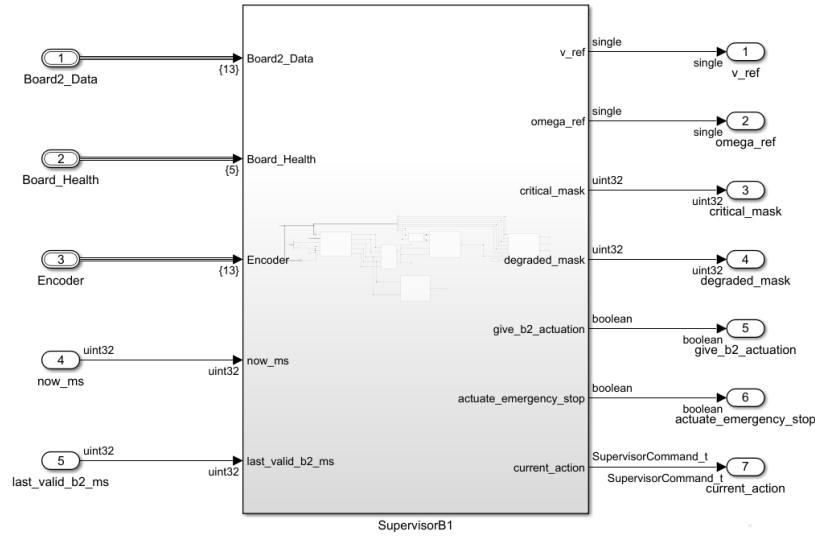


Figure 7: Schema a blocchi del modulo SupervisorB1.

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.

### 3.2 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 8: 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 9: Struttura dati BoardHealth.

- **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 10: 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.

### 3.3 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.



Table 2: Stati degradato e critico per Board 1

| OUTPUT             | VALORE               | SIGNIFICATO  |
|--------------------|----------------------|--|
| <b>temp_status</b> | TEMP_HEALTH_DEGRADED | Quando la temperatura è nel range<br>$] - 15; -5] \cup [55; +60[$  |
|                    | TEMP_HEALTH_CRITICAL | <ul style="list-style-type: none"> <li>• Quando la temperatura è per almeno 4s nel range<br/> <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> |
|                    | 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 5s nel range <math>[0\%; 15\%]</math></li> <li>• Nel caso in cui l'intervallo di tempo dall'ultimo aggiornamento della percentuale batteria è di 0.5s, 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 $> 25\%$   |

| OUTPUT                 | VALORE                 | SIGNIFICATO   |
|------------------------|------------------------|---|
| <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. |

#### 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 *isMotionConsistent*, indica che il sistema di fallback non è più in grado di garantire una stima affidabile del moto.

| OUTPUT               | VALORE       | SIGNIFICATO  |
|----------------------|--------------|--|
|                      | 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> ) |
| <b>b2_sup_status</b> | 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.  |

| OUTPUT    | VALORE      | SIGNIFICATO  |
|-----------|-------------|--|
|           | SUP_OK      | Quando non ci sono ne condizioni critiche ne degradate imposta questo valore.  |
| 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.  |

### 3.5 Decidere di far comandare la Board2

Il sottosistema per decidere se autorizzare o meno la Board2 a muovere il rover è mostrato in Figura 12.

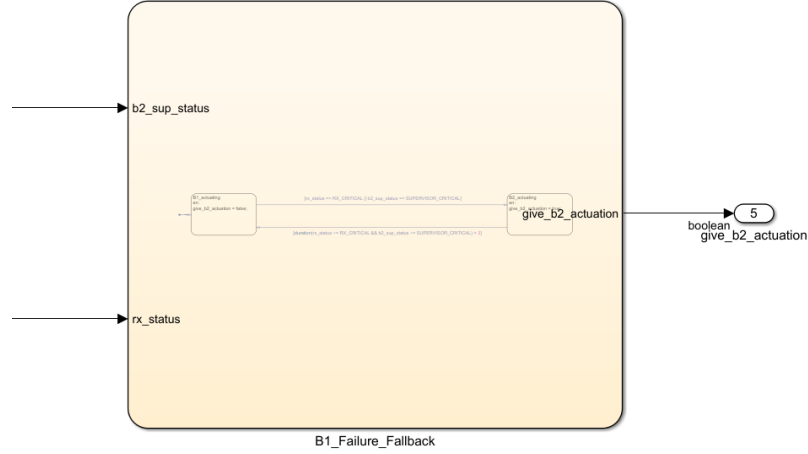


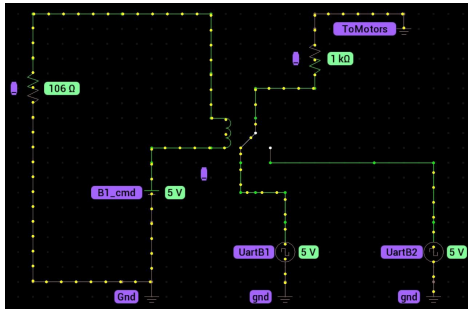
Figure 12: Chart per decidere se autorizzare o meno la Board2 a muovere il rover.

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

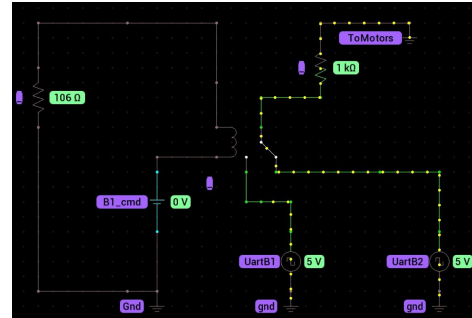
- **give\_b2\_actuation = 1:** Il relè è aperto e il movimento viene gestito da Board 2. Questa condizione si ha quando sono riscontrate anomalie *critiche* in ricezione (*texttrx\_status* = *RX\_CRITICAL*), *supervisore* (*textitb2\_sup\_status* = *SUP\_CRITICAL*).
- **give\_b2\_actuation = 0:** Il 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 13b e 13a.





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



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

### 3.6 Costruzione maschere degradate e critiche

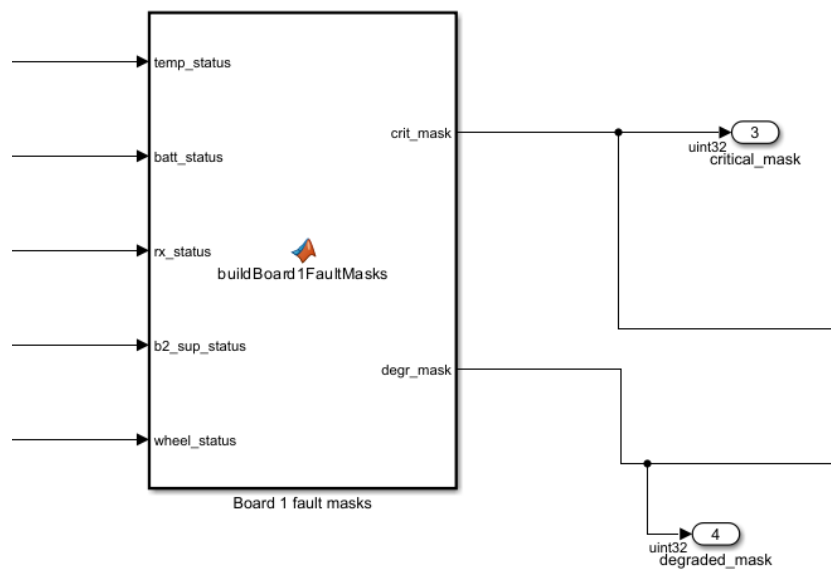


Figure 14: Chart per la costruzione delle maschere di errore critiche e degradate.

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

### 3.7 Calcolo riferimenti

Il sottomodello per il calcolo dei riferimenti di velocità lineare e angolare del rover è mostrato in Figura 15.

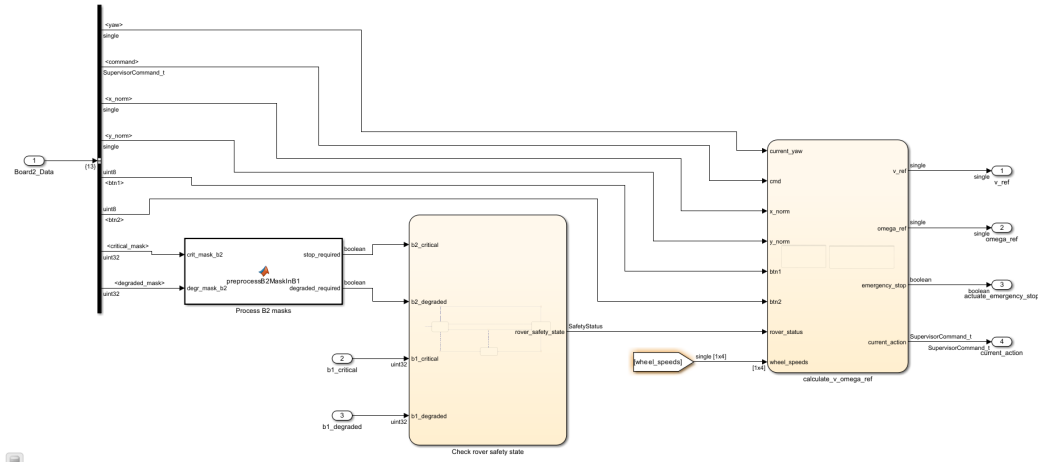


Figure 15: Chart per il calcolo dei riferimenti di velocità lineare e angolare del rover.

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 16.

```
function [V_MAX, OMEGA_MAX, V_MAX_MANEUVER, OMEGA_GO_LEFT, OMEGA_GO_RIGHT] = ...
    updateSafetyLimits(rovers_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 rovers_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 rovers_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 16: Scaling dei riferimenti in base alla gravità dell'anomalia rilevata.

Nella figura 16 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 17.

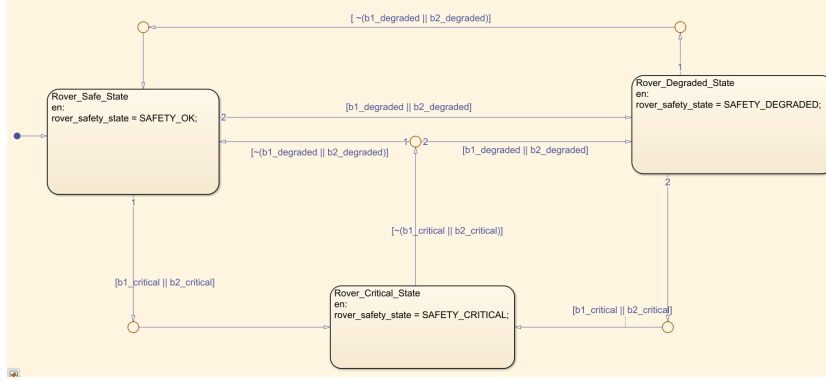


Figure 17: Calcolo della variabile safety\_status a partire dalle maschere di errore critiche e degradate di entrambe le Board.

## 4 Supervisore Board 2

### 4.1 Panoramica generale

Il supervisore della Board 2 è implementato come un modulo Simulink denominato **SupervisorB2**, il cui schema a blocchi è illustrato in Figura 18.

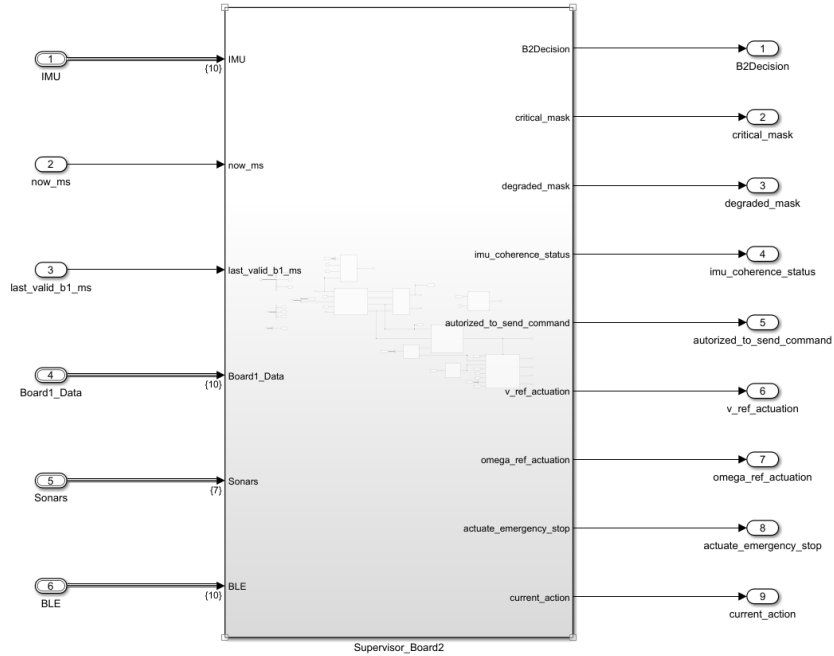


Figure 18: Schema a blocchi del modulo SupervisorB2.

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.

## 4.2 Input

I segnali in input che riceve sono:

## 4.3 Output

I segnali in output che fornisce sono:

## 4.4 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.

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

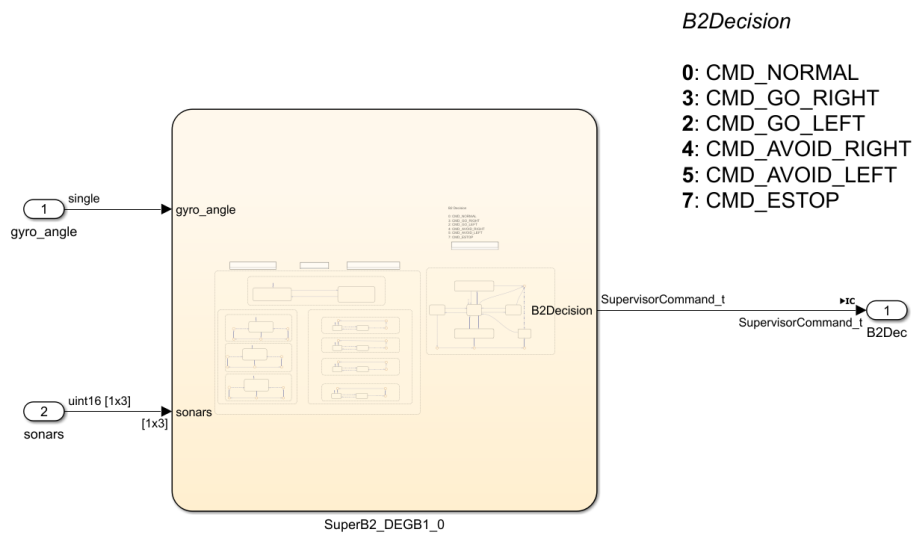


Figure 19: Logica di gestione degli ostacoli in stato non degradato.

Il chart per la gestione degli ostacoli in stato non degradato è mostrato in Figura 20.

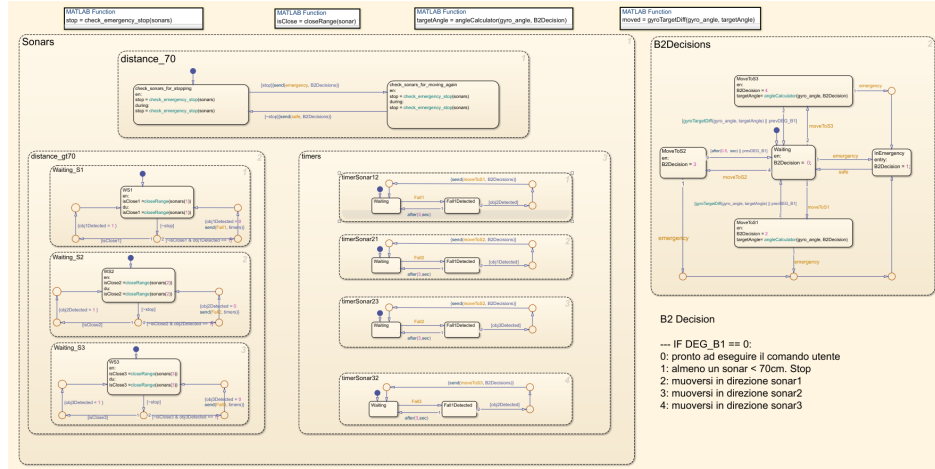
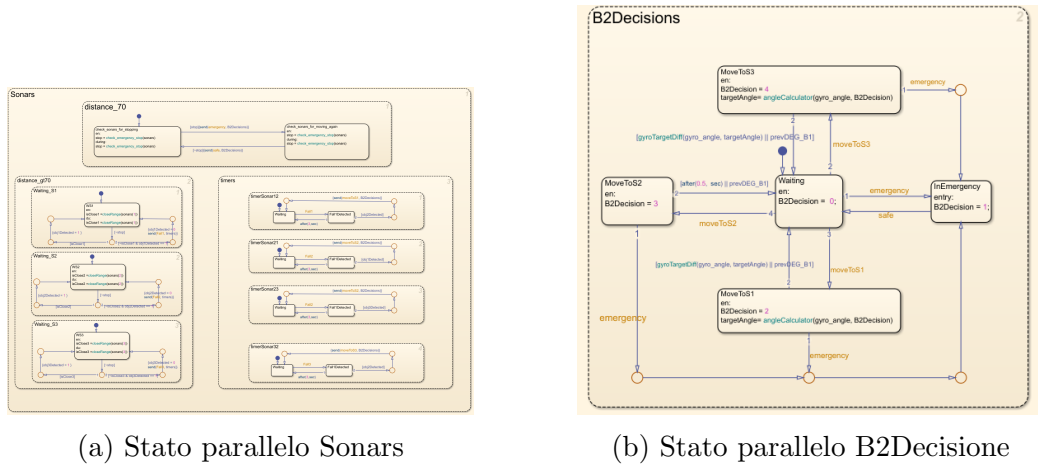


Figure 20: Chart di gestione ostacoli in stato non degradato.

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



(a) Stato parallelo Sonars

(b) Stato parallelo B2Decisione

Figure 21: Stati paralleli del chart in stato non degradato.

1. **B2Decisions:** Lo stato parallelo *B2Decisione* è dipendente dallo stato *Sonars* in quanto le sue transizioni vengono attivate da segnali provenienti da *Sonars*. In base ai segnali ricevuti, è capace di settare la variabile di output del chart, variabile che indica la decisione presa dal supervisore. Quindi, in questo stato si determina l'output del chart, che è un numero che varia da 0 a 4. Le azioni possibili includono l'arresto immediato del rover o la deviazione del percorso in base alla posizione dell'ostacolo. In quest'ultimo caso, la deviazione dura fintanto che il

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.

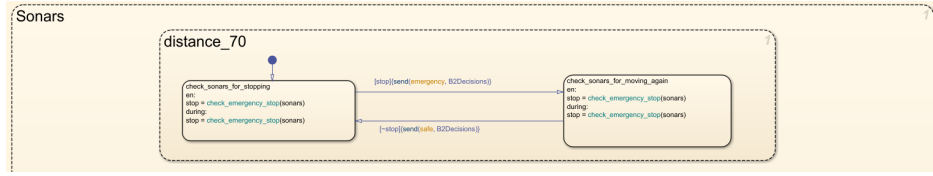


Figure 22: Stato parallelo per la gestione di un ostacolo a distanza  $\leq 70$  cm.

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)



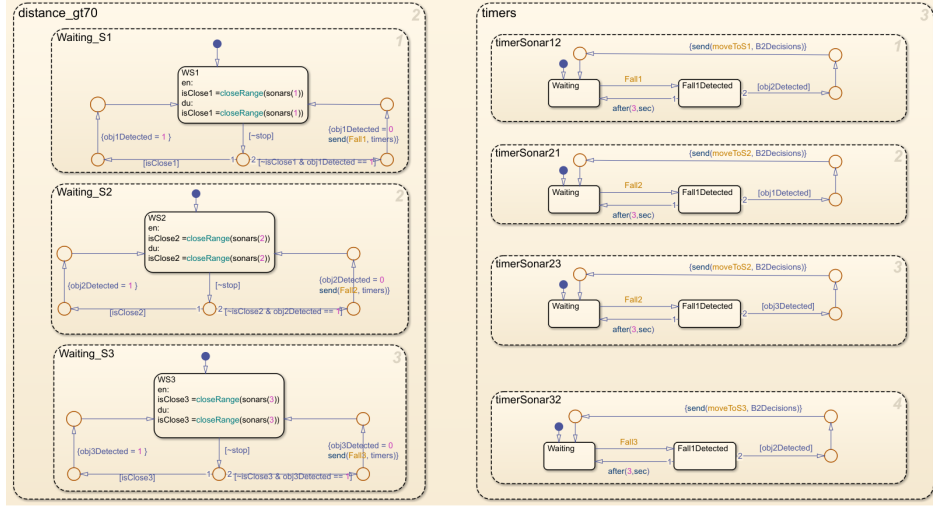


Figure 23: Stato parallelo per la gestione di un ostacolo in movimento a distanza  $> 70$  cm.

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.

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

Il chart per la gestione degli ostacoli in stato degradato è mostrato in Figura 24.

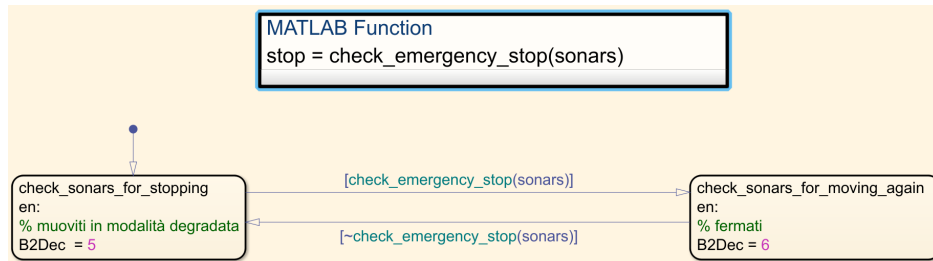


Figure 24: Chart di gestione ostacoli in stato degradato.

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.