

# 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>Supervisore Board 2</b>	<b>5</b>
2.1	Rilevamento ostacoli . . . . .	5
2.1.1	Gestione ostacoli con sistema in stato <i>nondegradato</i> . .	6
2.1.2	Gestione ostacoli con sistema in stato <i>degradato</i> . . . .	9
<b>3</b>	<b>Supervisore Board 1</b>	<b>9</b>
3.1	Panoramica generale – Input e Output . . . . .	9
3.1.1	Input . . . . .	10
3.1.2	Output . . . . .	11
3.2	Nuova subsection . . . . .	12

## List of Figures

1	Disposizione dei sensori sulla Board 2. . . . .	3
2	Segnale generato dal sensore HC-SR04 in presenza di un ostacolo a 3 metri di distanza. . . . .	3
3	Configurazione del DMA per la lettura dei segnali dai sensori. . . . .	4
4	Callback eseguita al termine della rilevazione dei fronti. . . . .	5
5	Chart di gestione ostacoli in stato non degradato. . . . .	6
6	Stati paralleli del chart in stato non degradato. . . . .	6
7	Stato parallelo per la gestione di un ostacolo a distanza $\leq 70$ cm. . . . .	7
8	Stato parallelo per la gestione di un ostacolo in movimento a distanza $> 70$ cm. . . . .	8
9	Chart di gestione ostacoli in stato degradato. . . . .	9
10	Schema a blocchi del modulo SupervisorB1. . . . .	9
11	Comandi in uscita dal supervisore della Board2. . . . .	10
12	Struttura dati Board_Health. . . . .	11
13	Struttura dati Encoder. . . . .	11

# 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.

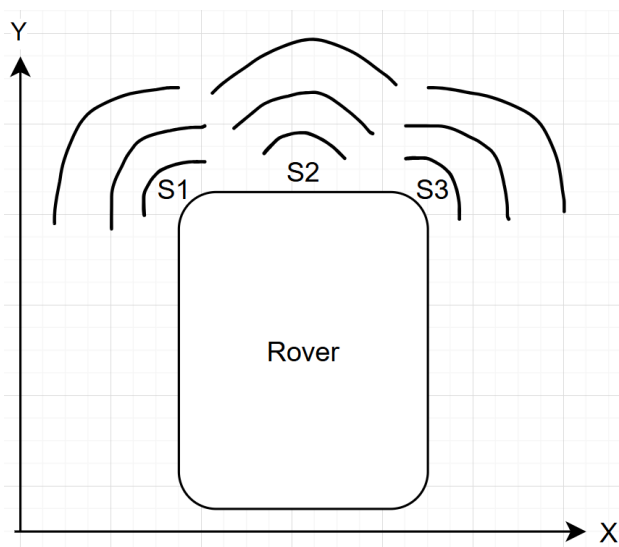


Figure 1: Disposizione dei sensori sulla Board 2.

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

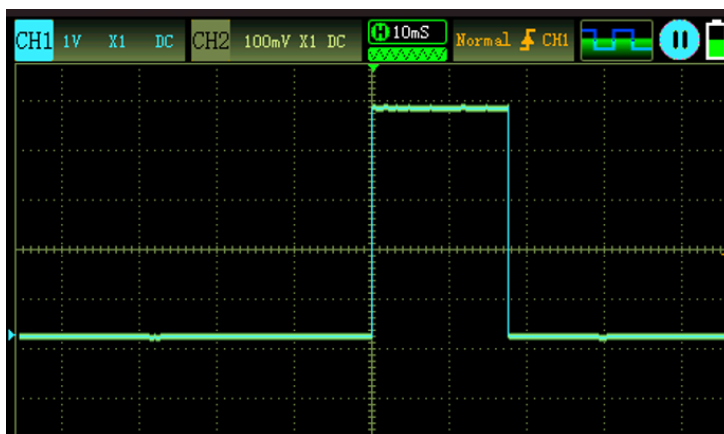


Figure 2: Segnale generato dal sensore HC-SR04 in presenza di un ostacolo a 3 metri di distanza.

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 Supervisore Board 2

### 2.1 Rilevamento ostacoli

Come da specifiche, il comportamento del rover, in presenza di ostacoli, deve essere funzione di due condizioni principali in cui il sistema 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 prima 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.

### 2.1.1 Gestione ostacoli con sistema in stato *nondegradato*

Il chart per la gestione degli ostacoli in stato non degradato è mostrato in Figura 5.

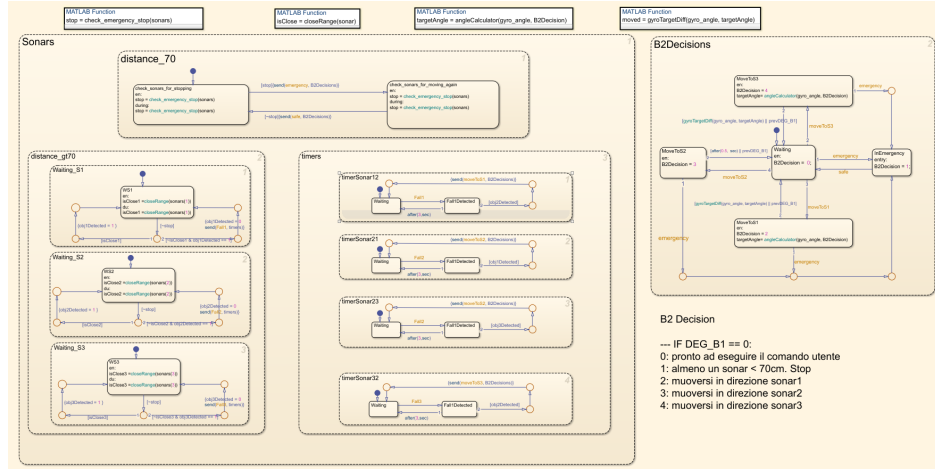
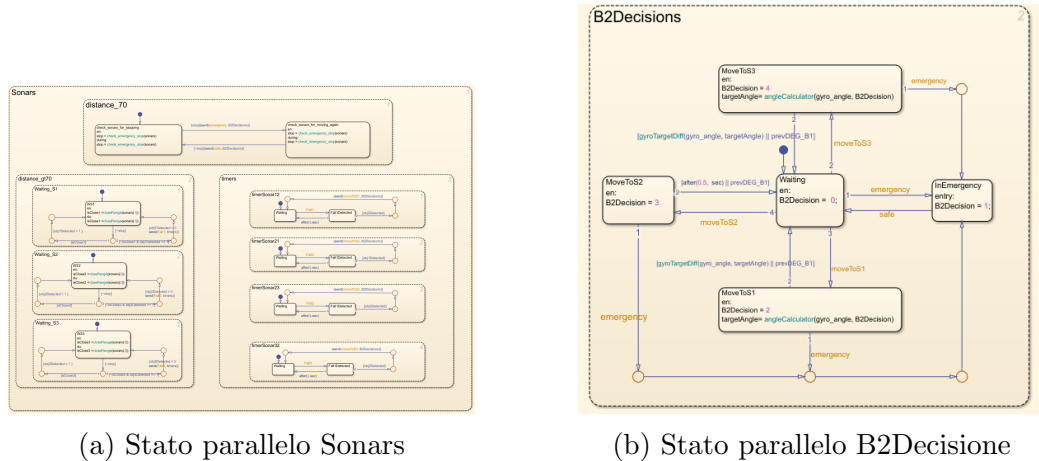


Figure 5: Chart di gestione ostacoli in stato non degradato.

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



(a) Stato parallelo Sonars

(b) Stato parallelo B2Decisione

Figure 6: 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.

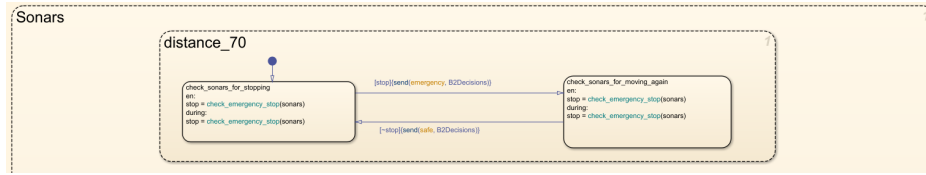


Figure 7: Stato parallelo per la gestione di un ostacolo a distanza  $\leq 70$  cm.

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)

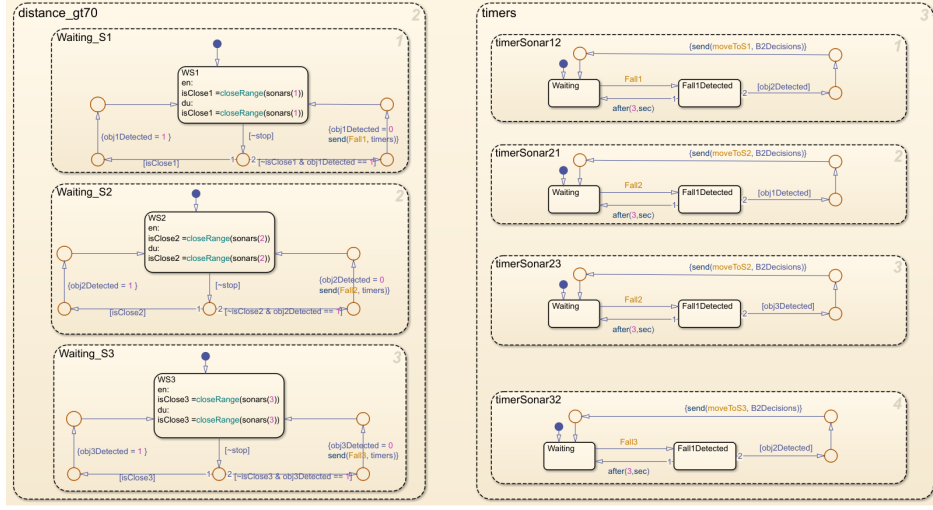


Figure 8: Stato parallelo per la gestione di un ostacolo in movimento a distanza  $> 70$  cm.

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.



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

Il chart per la gestione degli ostacoli in stato degradato è mostrato in Figura 9.

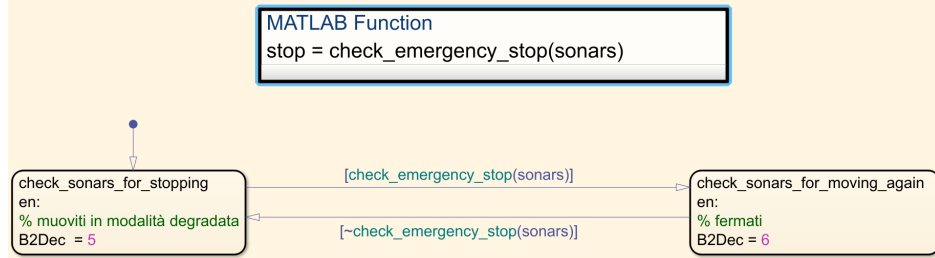


Figure 9: Chart di gestione ostacoli in stato degradato.

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.

## 3 Supervisore Board 1

### 3.1 Panoramica generale – Input e Output

Il supervisore della Board 1 è implementato come un modulo Simulink denominato **SupervisorB1**, il cui schema a blocchi è illustrato in Figura 10.

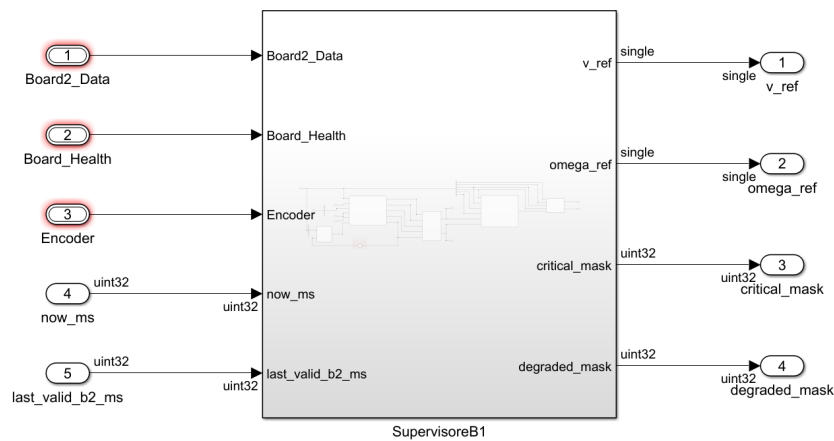


Figure 10: Schema a blocchi del modulo SupervisorB1.

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.

### 3.1.1 Input

I segnali in input che riceve sono:

- **Board2\_Data:** rappresenta i dati provenienti dalla Board2. Quelli utilizzati dal supervisore 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 11: Comandi in uscita dal supervisore della Board2.

- *x\_norm* & *y\_norm*: rappresenta il comando utente proveniente dal joystick.
- *yaw*: rappresenta l'angolo di orientamento del rover, calcolato a partire dai dati provenienti dalla Board2.
- *imu\_coherence\_status*: rappresenta la coerenza dei dati ricevuti dal sensore IMU, utile a rilevare motor fault.
- *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.
- **Board\_Health:** rappresenta lo stato della Board1. La struttura dati è la seguente:

```

typedef struct
{
    float temperature_degC;
    float battery_pct;

    uint32_t task_last_run_ms;    /* ultima esecuzione del task */

    uint32_t temp_last_valid_ms;
    uint32_t batt_last_valid_ms;
} BoardHealthSnapshot_t;

```

Figure 12: Struttura dati Board\_Health.

- **Encoder:** rappresenta i dati provenienti dagli encoder delle ruote del rover. 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 13: Struttura dati Encoder.

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

### 3.1.2 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	Errore termico (Logica predittiva)
1	BATT_CRI/DEG	Tensione batteria sotto soglia minima
2	COMM_CRI/DEG	Errore ricezione Board-to-Board
3-6	WHEEL_CRI/DEG	Guasti attuatori (FL, FR, RL, RR)
7	SUP_CRI/DEG	Timeout comunicazione Supervisore B2

Table 1: Mappatura della maschera di errore a 8 bit

### 3.2 Nuova subsection