Veicolo auto guidato

Con rilevamento di ostacoli e selezione del percorso

Sommario

Leonardo Ligabue
Università di Pisa

Leonardo Ligabue
Università di Pisa

1 Obiettivo

L'obiettivo di questo progetto è l'implementazione di un veicolo 4WD autoguidato in 2 scenari distinti:

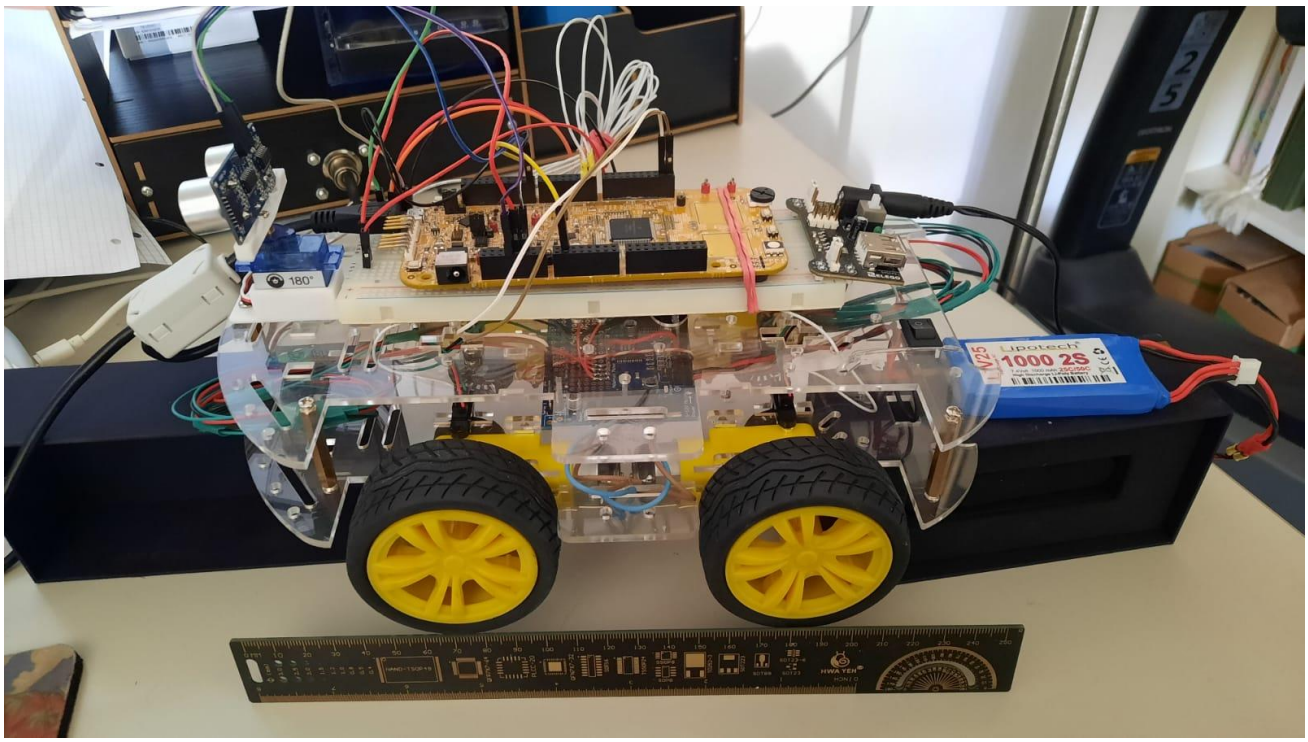
1. Raggiungimento di una destinazione su una mappa nota con selezione di direzioni preferenziali in funzione della posizione relativa della macchina rispetto alla destinazione
2. Raggiungimento di una destinazione su una mappa vuota usando un SONAR (e/o un LIDAR) per aggiornare la mappa con eventuali ostacoli e selezionare il percorso di conseguenza con la metodologia dello scenario 1

Il sistema finale deve quindi realizzare un interfacciamento fra il microcontrollore e le varie periferiche a disposizione:

1. 2x Switch
2. LED RGB
3. Servomotore
4. Modulo SONAR ToF
5. Modulo LIDAR UART
6. Driver motori I2C
    a. 4x Motori DC
7. 4x Encoder ottici

Inoltre, è prevista la capacità del veicolo di comunicare la mappa via UART a terminale su PC in modo da verificare il percorso seguito nello scenario operativo 1 ma anche ricevere gli update degli ostacoli effettuati nello scenario operativo 2.

Nella seguente foto è visibile il setup utilizzato per le prove:



Leonardo Ligabue
Università di Pisa

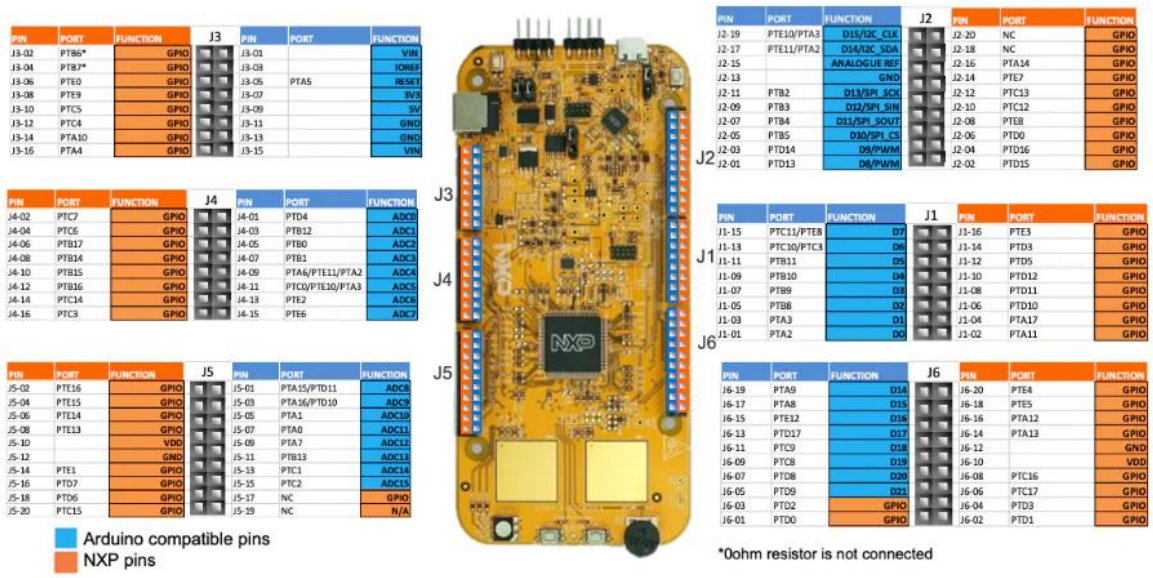
2 Hardware

2.1 Elenco componenti

- S32K144EVB NXP Evaluation Board
- HC-SR04 Ultrasonic Sensor
- KIT MOTORSHIELD ARDUINO V2 KIT2.3 Motor driver
- SER0047 Servomotore
- OPB880T55Z Infrared sensor
- 110090263 Kit motori/chassis/route Seeed Technology
- LTV25 Batteria al litio 7.4V 1000mAh
- 545043 Elegoo 5V DC-DC
- AC-DC 9V output
- Cavo USB a micro USB
- Breadboard
- 3D printed servo holder
- 3D printed servo to ultrasonic holder
- Fili e jumper

2.2 Connessioni

Nell'immagine di seguito è presente lo schema di connessione della board S32K144EVB



2.2.1 Connessioni alimentazione

2.2.1.1 5V

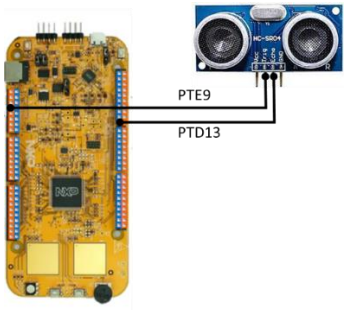
Tutti i GND sono connessi insieme, la board S32K144EVB ricava l'alimentazione dall'USB e alimenta tramite J3-09 l'I2C. Ogni altra alimentazione viene presa dalla board 545043 Elegoo. Le connessioni vengono effettuate con l'ausilio di una breadboard.

2.2.1.2 7.4V

La batteria al litio alimenta l'uscita del driver motore attraverso un interruttore basculante per attivare o disattivare il funzionamento dei motori.

Leonardo Ligabue
Università di Pisa

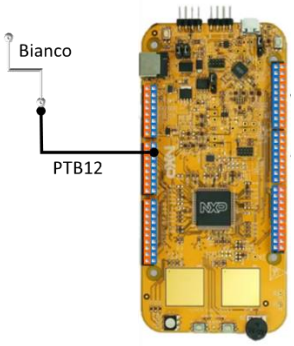
2.2.2 Connessioni HC-SR04



2.2.3 Connessioni servomotore

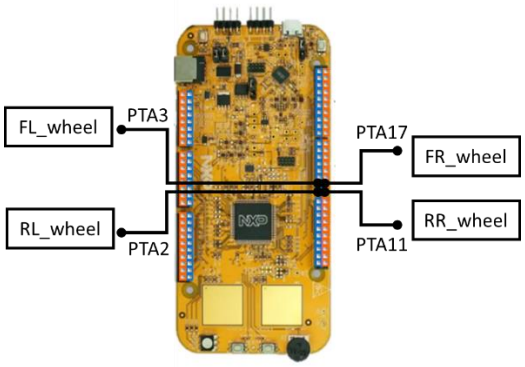
Colore fili:

- Bianco → PWM
- Rosso → Vcc
- Nero → GND
- Marrone → Feedback

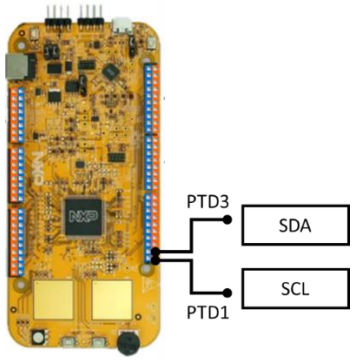


2.2.4 Connessioni encoders

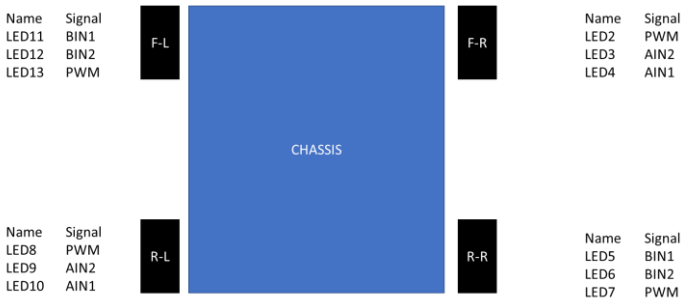
La parte di invio del LED dell'encoder è stata preventivamente realizzata con un pull-up per ogni encoder sulla board del driver motore. Le connessioni lato board S32K144EVB dei fili bianchi di output dei sensori sono state effettuate come da immagine seguente:



2.2.5 Connessione I2C

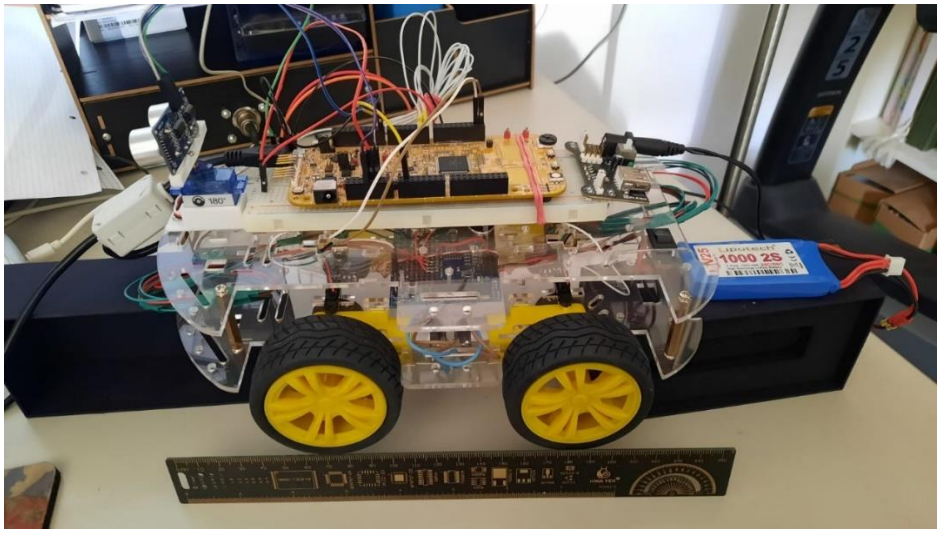


2.2.5.1 Connessione motori a driver



2.3 Setup di prova

Il setup utilizzato è il seguente:



3 Approccio

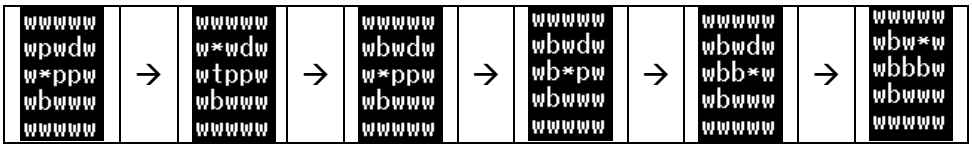
L'approccio globalmente utilizzato per la realizzazione finale non può essere di tipo PiL a causa della presenza di interrupt sui GPIO che sono incompatibili con il build del codice per l'utilizzo in tale modalità (come da problema riscontrato durante il corso). Pertanto, l'implementazione è stata di tipo "try and error" sfruttando la comunicazione tramite UART di eventuali valori necessari per la valutazione del corretto funzionamento delle funzioni specifiche realizzate.

Per semplicità di progettazione il movimento è previsto all'interno di una mappa realizzata con una griglia 5x5 (implementata a livello di codice con un vettore di 25 elementi per conformità con i tipi utilizzabili in simulink) considerando ogni cella con un lato di 50 cm. Ad ogni cella della griglia viene associato un carattere per identificare la tipologia di elemento del percorso:

1. w → Ostacolo
2. p → Nuovo percorso
3. t → Percorso già attraversato
4. b → Percorso bloccato
5. d → Destinazione
6. * → Posizione attuale

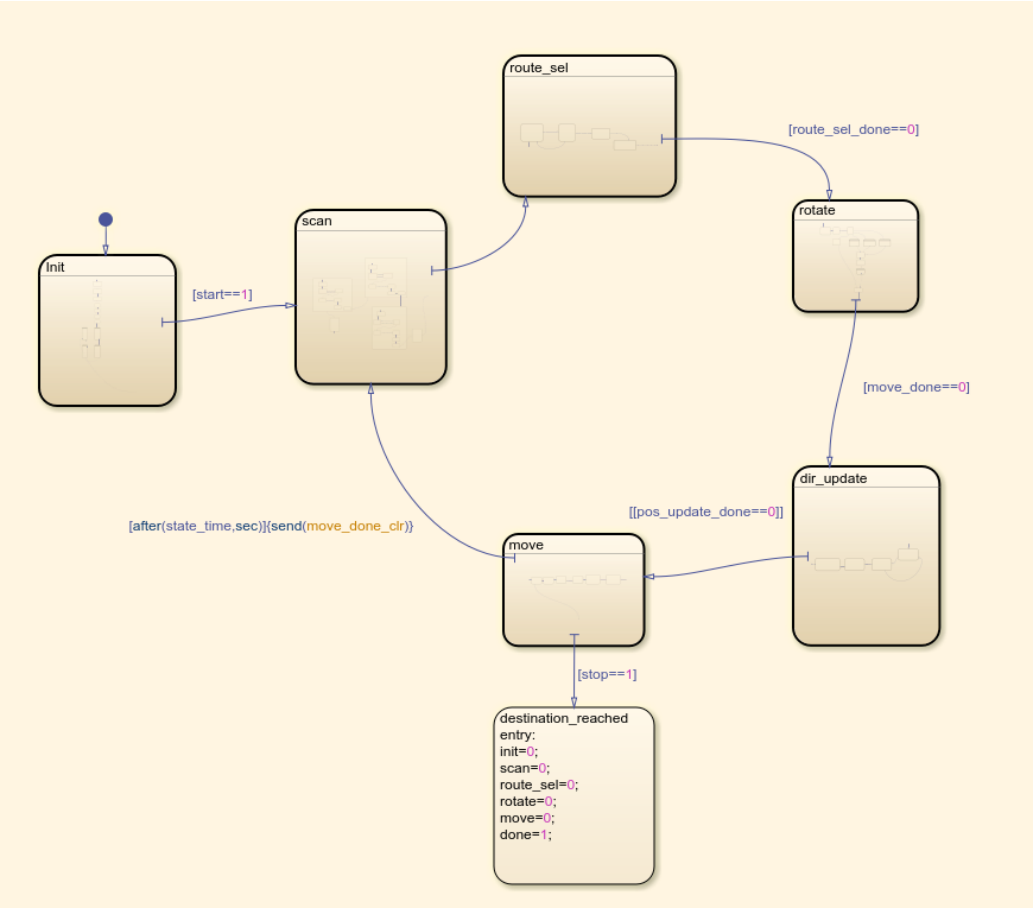
Simulink non gestisce i char come tipo di variabile, pertanto è necessario utilizzare degli uint8 corrispondenti alla codifica ASCII di ogni carattere che si vuole utilizzare in modo da ottenere una visualizzazione della mappa coerente a quanto deciso in termini di nomenclatura delle tipologie di celle.

L'immagine seguente indica un esempio di funzionamento in una mappa preimpostata comunicata tramite UART a terminale su PC ad ogni ciclo:



NOTA: La dimensione effettivamente utilizzabile della mappa in realtà è la sottogriglia 3x3 interna, in quanto per come viene gestito il codice di selezione del percorso e della gestione delle scansioni, il caso limite delle posizioni di bordo mappa non viene trattato.

L'intero progetto è stato realizzato come modello SIMULINK e implementato principalmente attraverso una macchina a stati e il lancio di trigger delle funzioni di gestione delle varie periferiche.



Dall'immagine è possibile identificare la sequenza operativa dei vari stati principali. Dopo una fase di inizializzazione preliminare il sistema cicla attraverso gli stati di scan, selezione del percorso, rotazione, update di direzione/posizione e movimento fintanto che la posizione corrente non corrisponde alla destinazione. Ogni stato è identificato da una specifica combinazione di colori del LED RGB.

3.1 Init

In questa fase vengono inizializzati:

1. Mappa:
   - Struttura griglia nota
   - Posizione iniziale
   - Direzione iniziale
   - Destinazione
2. Duty Cycle del servomotore in posizione frontale
3. Valore di confronto del counter degli encoder
4. Motori attraverso comunicazione I2C con il driver

3.2 Scan

La fase di scansione dei dintorni avviene in 3 fasi principali:

1. Scan_left → Scansione a sinistra rispetto alla direzione attuale
2. Scan_front → Scansione frontale rispetto alla direzione attuale
3. Scan_right → Scansione a destra rispetto alla direzione attuale

In ogni scansione viene effettuato il trigger delle misure di SONAR (e/o LIDAR) e viene poi effettuata l'elaborazione della mappa in funzione dei risultati ottenuti al fine di modificare in ostacoli, quando necessario, gli slot della griglia scansionati, permettendo una evoluzione dinamica della mappa.

La misura del SONAR avviene fornendo un trigger di 10 us sull'apposito pin. Il segnale di ritorno sul pin ECHO del modulo ha un valore alto per una durata proporzionale alla distanza del bersaglio. In particolare, la distanza si ricava come:

$$SONAR_{dist}[cm] \cong \frac{Durata\ ECHO\ alto\ [\mu s]}{2 * 29\ [\frac{\mu s}{cm}]}$$

Il fattore 2 dipende dal fatto che il segnale acustico del SONAR percorre la distanza fra sorgente e bersaglio in avanti e poi indietro. Il fattore 29 deriva dalla considerazione che la velocità del suono nell'aria vale:

$$v = 343\ [\frac{m}{s}]$$

Che consideriamo fissa ed indipendente dalla temperatura per semplicità di calcolo senza intaccare l'efficacia del sistema in quanto non necessità di una precisione elevata nella misura della distanza. Possiamo a questo punto ricavare il fattore 29 come:

$$29 \cong \frac{1}{343 * \frac{100}{10^6}}$$

Se la distanza misurata dal SONAR risulta inferiore ad una certa soglia, significa che nella cella adiacente scansionata è presente un ostacolo e quindi la mappa viene aggiornata di conseguenza.

3.3 Route_sel

La selezione del percorso avviene valutando la direzione preferenziale da seguire in funzione della distanza relativa fra posizione corrente e destinazione. L'ordine di preferenza, a parità di condizioni, è il seguente:

SU > DESTRA > Giù > SINISTRA

Inoltre, un altro layer di preferenza con priorità maggiore è il seguente:

Nuovi_percorsi > Percorsi_già_attraversati

Ad esempio, supponendo che la direzione preferita sia SU, se esiste un nuovo percorso con direzione SU, sarà selezionato, altrimenti prima valuto con l'ordine SU > DESTRA > Giù > SINISTRA i nuovi percorsi, altrimenti reitero questa selezione sui percorsi già attraversati dando sempre priorità al percorso preferito oppure all'ordine predefinito SU > DESTRA > Giù > SINISTRA se non fosse possibile intraprendere il percorso preferenziale.

3.4 Rotate

La rotazione dipende dalla posizione relativa fra direzione corrente e direzione target e viene rappresentata con un parametro chiamato "rot":

- Rot = 0 → Nessuna rotazione
- Rot = 1 → Rotazione di 90° a sinistra
- Rot = 2 → Rotazione di 90° a destra
- Rot = 3 → Rotazione di 180° a destra

Per effettuare una rotazione le ruote del veicolo devono essere pilotate con verso di rotazione opportuno:

$$Rotazione\ a\ sinistra \rightarrow \begin{cases} Ruote\ di\ sinistra = \ indietro \\ Ruote\ di\ destra = \ avanti \end{cases}$$

$$Rotazione\ a\ destra \rightarrow \begin{cases} Ruote\ di\ sinistra = \ avanti \\ Ruote\ di\ destra = \ indietro \end{cases}$$

Per semplicità di calcolo è stata considerata solo una coppia di ruote per il calcolo dello spazio da percorrere per effettuare una rotazione. In particolare, i parametri usati per il calcolo sono:

$$R = 13.5\ cm \rightarrow distanza\ fra\ 2\ ruote\ sullo\ stesso\ asse$$

$$D = 6.3\ cm \rightarrow diametro\ di\ una\ ruota$$

Durante la rotazione le ruote si muovono lungo una circonferenza di raggio R/2 e devono quindi percorrere una distanza pari a un quarto del perimetro:

$$dist_{90^\circ} = \frac{R}{2} * 2 * \pi * \frac{1}{4} = \frac{R\pi}{4}$$

Ogni encoder è composto da 40 divisioni, per cui ogni sua commutazione corrisponde ad un quarantesimo del perimetro della ruota in termini di spazio percorso. Per cui, se considero N = numero di letture di encoder:

$$dist_{encoder} = \frac{D\pi}{40} * N$$

Posso ricavare quindi il valore di N da comparare ai counter degli encoder di ogni singola ruota come:

$$dist_{90^\circ} = dist_{encoder} \rightarrow \frac{R\pi}{4} = \frac{D\pi}{40} * N \rightarrow N = \frac{40}{4} * \frac{R\pi}{D\pi} = 10\frac{R}{D} \cong 21$$

Il valore per una rotazione di 180° viene quindi considerato banalmente 42.

3.5 Dir_update

La direzione e la posizione correnti vengono aggiornate con quelle target

3.6 Move

Viene attivato l'output dei 4 motori e con calcoli analoghi a quelli della rotazione viene stabilito il compare degli encoder a 100 per effettuare uno spostamento di 50 cm. (Ogni lettura di encoder corrisponde a circa 0.5 cm)

3.7 Destination_reached

Se la posizione target corrisponde con la destinazione significa che dopo il movimento il programma deve arrestarsi

4 Firmware

4.1 Blocchi di configurazione

4.1.1 Configurazione interrupt e pull-up dei GPIO:

- PTC13 fronte in salita →
  SW3 Utilizzato per inviare via UART una copia della mappa attuale settando il flag **"meas_done"**
- PTC12 fronte in salita →
  SW2 Utilizzato per far partire il veicolo una volta completata la fase di inizializzazione. Setta il flag **"start"**
- PTD13 entrambi i fronti →
  Pin dell'ECHO del sonar, su rising edge inizio a contare la durata del segnale settando il flag **"us_echo_mode"** e su falling edge smetto resettando il flag **"us_echo_mode"** e settando il flag **"SONAR_scan_done"**.
- PTA2 entrambi i fronti →
  Encoder ruota posteriore sinistra. Incremento il relativo counter ad ogni commutazione del pin. Effettuo una lettura in **"aux_pullup_2"** con priorità maggiore rispetto alla gestione dell'ISR al fine di attivare il pull-up del GPIO.
- PTA3 entrambi i fronti →
  Encoder ruota anteriore sinistra. Incremento il relativo counter ad ogni commutazione del pin. Effettuo una lettura in **"aux_pullup_1"** con priorità maggiore rispetto alla gestione dell'ISR al fine di attivare il pull-up del GPIO.
- PTA11 entrambi i fronti →
  Encoder ruota posteriore destra. Incremento il relativo counter ad ogni commutazione del pin. Effettuo una lettura in **"aux_pullup_3"** con priorità maggiore rispetto alla gestione dell'ISR al fine di attivare il pull-up del GPIO.
- PTA17 entrambi i fronti →
  Encoder ruota anteriore destra. Incremento il relativo counter ad ogni commutazione del pin. Effettuo una lettura in **"aux_pullup_4"** con priorità maggiore rispetto alla gestione dell'ISR al fine di attivare il pull-up del GPIO.

4.1.2 Configurazione LPTMR

Questo timer viene utilizzato con un interrupt settato a 10 us.

Il suo funzionamento dipende dal flag attualmente a livello alto fra:

- **"us_trig_mode"** →
  Porta a livello basso il pin PTE9 collegato al pin TRIGGER del modulo sonar e resetta il flag **"us_trig_mode"**.
- **"us_echo_mode"** →
  Incrementa il counter **"period"** ad ogni interrupt contando la durata del livello alto del segnale di ECHO del modulo sonar.

4.1.3 Configurazione I2C

Utilizzando il FlexIO I2C master config:

- Istance number: 0
- Mode: Polling
- Baud Rate: 10000

Utilizzando i seguenti pin per il BUS I2C:

- PTD3 → SDA
- PTD1 → SCL

L'indirizzo del driver motori è 96 in decimale.

4.1.4 Configurazione PWM

Utilizzando il modulo FTM

- Enable del modulo 0
- Edge-Aligned PWM
- Frequenza: 50 Hz
- Deadtime: 0 ns

Utilizzando il pin di output del Canale 0: **PTB12**

Il duty cycle del PWM viene settato attraverso la variabile **"servo_pwm_duty"**.

4.1.5 Configurazione LPUART

Configurata su istanza:1 per comunicare attraverso openSDA

4.2 Blocco LED

Il LED RGB della scheda EVB142 viene utilizzato per identificare lo stato di funzionamento attuale del dispositivo attraverso specifici flag settati nella macchina a stati:

- Init → Blu
- Scan → Rosso
- Route_select → Verde
- Rotate → Viola (Blu+Rosso)
- Move → Arancione (Rosso + Verde)
- Done → Bianco (Blu + Rosso + Verde)

4.3 Blocco gestione movimento

Se il flag **"move_done"** non è settato allora eseguo un compare del valore presente nei counter degli encoder delle ruote con la soglia selezionata in dipendenza di movimento o tipo di rotazione. Se almeno uno dei counter supera la soglia allora setto il valore di **"move_done"**. Se tale flag è settato allora azzero il valore dei counter degli encoder delle ruote per i movimenti successivi. Viene utilizzata una OR dei compare dei valori degli encoder in quanto i vari motori non sono perfettamente tarati e inoltre, durante la rotazione, essendo il veicolo 4WD c'è la possibilità che le ruote possano slittare e quindi che i relativi encoder non contino. I counter specifici di ogni singola ruota sono:

- RL_wheel_counter (PTA2) → encoder ruota posteriore sinistra (rear left)
- FL_wheel_counter (PTA3) → encoder ruota anteriore sinistra (front left)
- RR_wheel_counter (PTA11) → encoder ruota posteriore destra (rear right)
- FR_wheel_counter (PTA17) → encoder ruota anteriore destra (front right)

E vengono incrementati ad ogni edge di commutazione sui relativi pin.

4.4 Blocco UART

Basato sul blocco di esempio della libreria dell'evaluation board con l'aggiunta del flag **"meas_done"** per iniziare la comunicazione e della funzione **map_uart_vectorization** per creare un vettore a partire dalla mappa al fine di visualizzarla come una matrice su terminale. Questo passaggio si ottiene scindendo in sotto-vettori di dimensione 5 la mappa e aggiungendo ad ogni sotto-vettore i valori ASCII di Line Feed [uint8(10)]

e Carriage Return [uint8(13)]. Inoltre, ad ogni linea successiva alla prima viene anche aggiunto un carattere di Spazio [uint(32)] per motivi di allineamento su terminale.

4.5 Macchina a stati

Le chart in Simulink, necessitano di avere ogni output definito in ogni fase del processo. Questo significa che non specificare il valore di un uscita in blocchi della chart ha come effetto la riscrittura ad ogni ciclo di programma del valore di uscita specifico indicato nei blocchi precedenti. Ad esempio, se il blocco1 dovesse scrivere un parametro x = 5, nel caso in cui la gestione x non fosse specificata nei blocchi successivi la chart continuerebbe a scrivere x = 5 ad ogni ciclo di programma. (O almeno è quello che è risultato dai miei test in fase di progettazione). Per questo motivo, vista la necessità di avere funzioni che devono essere lanciate una singola volta e per avere la certezza che le funzioni vengano completamente svolte prima del passaggio allo stato successivo, la chart utilizzata fa largo uso dei comandi send(trigger) per il lancio delle funzioni specifiche e dei clear dei flag utilizzati.

4.5.1 Init chart

Viene lanciata l'inizializzazione della mappa con il trigger **"init_trigger"**. Sulla mappa vengono inseriti i valori seguenti:

- Posizione iniziale
- Direzione iniziale
- Destinazione
- Mappa iniziale (La mappa iniziale è completamente riempita di "p" nella modalità operativa 2)

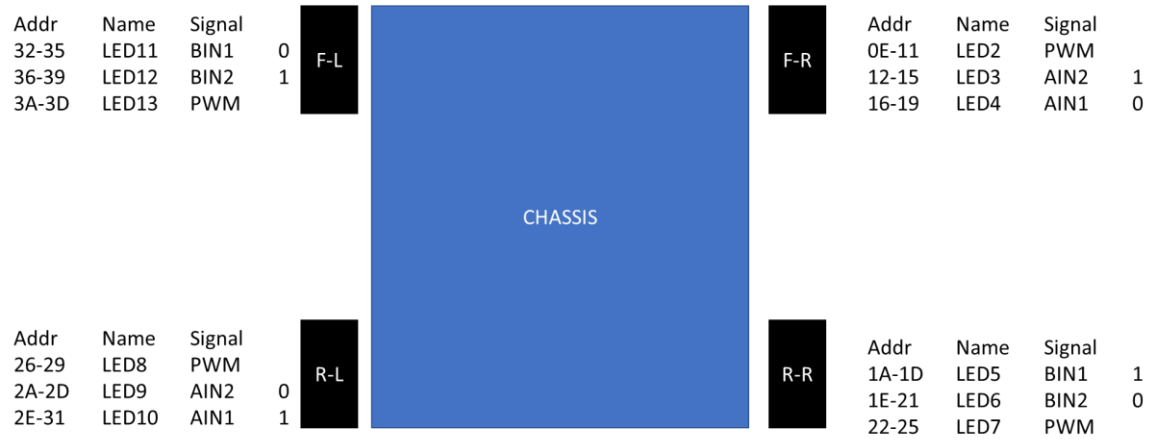
Viene inizializzato il servomotore in posizione frontale settando il duty cycle a 0.075.

Viene inizializzato il valore del compare dei counter di movimento a 100 per evitare che nella fase di movimento si verifichino falsi positivi nella comparazione e il movimento non avvenga.

Viene inizializzato il driver per il movimento delle 4 ruote con comunicazione I2C per settare i valori del chip **TB6612** attraverso la scrittura sul chip **PCA9685**. Preliminarmente vengono settati i valori di MODE1 e MODE2:

- MODE1
  - Address = 0
  - Data = 1
- MODE2
  - Address = 0
  - Data = 0 (Per configurare l'uscita in modalità open-drain e quindi non pilotare i motori)

Sulla motor driver board i collegamenti alle uscite dei motori sono strutturati come da immagine seguente, considerando gli indirizzi dei registri in formato esadecimale (nel modello simulink vengono usati i corrispondenti valori in decimale di tipo uint8 per conformità al blocco I2C della libreria NXP):



I valori settabili sono composti da 4 registri ciascuno, fare riferimento al datasheet del chip **PCA9685** per la modalità con cui settare i corretti valori all'interno dei 4 registri di ogni uscita LED#. La comunicazione avviene settando i parametri di indirizzo e data da utilizzare e lanciando il trigger di comunicazione I2C. Il programma resta in uno stato di attesa per un tempo **"state_time"** al fine di garantire la non sovrapposizione delle comunicazioni.

Una volta completata l'inizializzazione il programma attende il settaggio del parametro "start" attraverso la pressione di SW2. Al passaggio di stato conseguente la pressione di SW2 viene inviata la mappa attraverso comunicazione UART a terminale su PC.

4.5.1.1 Codice init map

```
function [map , initial_pos, initial_dir, init_destination, init_map_done,
start]  = fcn(init_map, prev_map, prev_pos, prev_dir, prev_start, destination)
% Example maps
%empty_map        =
uint8([112,112,112,112,112,112,112,112,112,112,112,112,112,112,112,112,112,1
12,112,112,112,112,112,112]);
% map_1           =
uint8([119,119,119,119,119,119,119,119,112,119,119,112,112,112,119,119,112,119,1
19,119,119,119,119,119,119]);
dead_end_map     =
uint8([119,119,119,119,119,119,112,119,112,119,119,112,112,112,119,119,112,119,1
19,119,119,119,119,119,119]);
map = prev_map;
init_destination = destination;
initial_pos = prev_pos;
initial_dir = prev_dir;
start = prev_start;
if(init_map)
    map = dead_end_map;
    init_destination = uint8(9);
    initial_pos = uint8(17);
    initial_dir = uint8(1);
    init_map_done = false;
    map(init_destination) = uint8(100);
    map(initial_pos)=uint8(42);
    start=false;
else
    init_map_done = init_map;
end
```

4.5.1.2 Codice I2C com

```
function  mex    = fcn(addr, data)
% The I2C message is vectorized as REGISTER_ADDRESS - DATA to be sent to
```

```
% the KIT MOTORSHIELD ARDUINO V2 KIT2.3 motor control driver
mex = [addr, data];
```

4.5.2 Scan chart

La scan chart è divisa in 3 fasi analoghe in cui viene settato il valore del duty cycle del servomotore più una fase di idle. Ogni fase è identificata dal parametro "scan_step" utilizzato nell'elaborazione della mappa:

- Idle → servo_PWM = 0.075 , scan_step = 0
- Scan left → servo_PWM = 0.125 , scan_step = 1
- Scan front → servo_PWM = 0.075 , scan_step = 2
- Scan right → servo_PWM = 0.025 , scan_step = 3

In ogni sottofase, dopo il settaggio del valore del duty cycle del PWM il programma attende un tempo "servo_move_delay" per permettere al servomotore di spostarsi correttamente in posizione. A questo punto vengono lanciati i trigger della misura del SONAR (reiterata dopo un tempo "timeout" nel caso in cui la misura del SONAR non vada a buon fine) e successivamente, della misura del LIDAR (attualmente in versione dummy). Il trigger della misura del SONAR effettua le seguenti operazioni:

- Resetta il counter "period" utilizzato per la misura della distanza
- Setta il valore del flag "us_trig_mode"
- Mette alto il valore del pin PTE9 per mandare al modulo HC-SR04 il trigger della misura sonar (che ricordo dover essere un segnale alto di 10 us)
- Successivamente viene avviato il timer LPTMR (configurato per contare ogni 10 us)

Dopo la conclusione delle misure di ogni fase viene lanciato il trigger per effettuare l'elaborazione della mappa. Le modifiche alla mappa vengono effettuate solo nel caso in cui il flag "scan_bypass" sia uguale a 0, questa condizione corrisponde alla modalità operativa 2, altrimenti, settando il valore di tale flag il programma svolge la modalità operativa 1.

4.5.2.1 Codice map elaboration

```
function [scan_left_done, scan_front_done, scan_right_done, map_update] =
fcn(scan_step, dir, map_in, SONAR_meas,LIDAR_meas, current_pos,scan_bypass)
% Sonar_meas is the number of ticks done every 10us when the ECHO is high
% level.
% So the high level time of the ECHO is SONAR_meas * 10 [us]
% SONAR_dist = SONAR_meas/58 [cm]
% The measure to define if a wall is present or not is done directly on the
% number of ticks counted in order to avoid problems with MATLAB variable
% types.
% Considering:
%   - A movement grid of 50 cm sided squares
%   - That the car is in the center of a grid square at any given step
% If the measure of the SONAR is less than 70 cm (instead of 75 in order to
% some error room) the adiacent square must be considered a wall
% Since 1 tick = 0.17 cm --> 70cm = 412 ticks
wall_ticks = 412;
% Values for current direction compares in the switch
dir_up      = uint8(1);
dir_right   = uint8(2);
dir_down    = uint8(3);
dir_left    = uint8(4);
% Values for scan_step compares in the switch
idle        = 0;
scan_left   = 1;
scan_front  = 2;
scan_right  = 3;
% Auxiliary map to comply with MATLAB assignment necessities inside
```

```
% functions
map_aux = map_in;
% Auxiliary scan done flags to comply with MATLAB assignment necessities
% inside functions
scan_left_done_aux  = 0;
scan_front_done_aux = 0;
scan_right_done_aux = 0;
% The switch case is done for every current direction of the car since the
% left, front and right scans will have different effects on the map update
% depending on the current direction the car is facing. This is because
% left, right and front for the scan refers to the direction relative to
% where the car is currently facing
switch dir
    case dir_up
        switch scan_step
            case scan_left
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos-1) = uint8(119);
                end
                scan_left_done_aux = 1;
            case scan_front
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos-5) = uint8(119);
                end
                scan_front_done_aux = 1;
            case scan_right
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos+1) = uint8(119);
                end
                scan_right_done_aux = 1;
            case idle
                scan_left_done_aux  = 0;
                scan_front_done_aux = 0;
                scan_right_done_aux = 0;
        end
    case dir_right
        switch scan_step
            case scan_left
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos-5) = uint8(119);
                end
                scan_left_done_aux = 1;
            case scan_front
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos+1) = uint8(119);
                end
                scan_front_done_aux = 1;
            case scan_right
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos+5) = uint8(119);
                end
                scan_right_done_aux = 1;
            case idle
                scan_left_done_aux  = 0;
                scan_front_done_aux = 0;
                scan_right_done_aux = 0;
        end
    case dir_down
        switch scan_step
            case scan_left
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
```

```
                    map_aux(current_pos+1) = uint8(119);
                end
                scan_left_done_aux = 1;
            case scan_front
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos+5) = uint8(119);
                end
                scan_front_done_aux = 1;
            case scan_right
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos-1) = uint8(119);
                end
                scan_right_done_aux = 1;
            case idle
                scan_left_done_aux  = 0;
                scan_front_done_aux = 0;
                scan_right_done_aux = 0;
        end
    case dir_left
        switch scan_step
            case scan_left
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos+5) = uint8(119);
                end
                scan_left_done_aux = 1;
            case scan_front
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos-1) = uint8(119);
                end
                scan_front_done_aux = 1;
            case scan_right
                if ((SONAR_meas < wall_ticks) & ~scan_bypass)
                    map_aux(current_pos-5) = uint8(119);
                end
                scan_right_done_aux = 1;
            case idle
                scan_left_done_aux  = 0;
                scan_front_done_aux = 0;
                scan_right_done_aux = 0;
        end
end
% Result assignments to the outputs
map_update = map_aux;
scan_left_done  = scan_left_done_aux;
scan_front_done = scan_front_done_aux;
scan_right_done = scan_right_done_aux;
```

4.5.3 Route_sel chart

Viene inviato il trigger della funzione della selezione del percorso. Il programma poi attende il completamento della selezione e successivamente lancia il trigger di clear del flag di selezione per le successive iterazioni del ciclo principale.

4.5.3.1 Codice selezione percorso

```
function [target_dir, target_pos, map_update, route_sel_done] = fcn( map_in,
current_pos, destination, current_dir)
% map character meaning:
%   w = wall
%   p = new_path
%   t = threaded path
%   b = blocked path
%   d = destination
w=  uint8(119); % Wall
p=  uint8(112); % New Path
t=  uint8(116); % Threaded Path
b=  uint8(98);  % Blocked Path
d=  uint8(100); % Destination
r=  uint8(114); % Reached destination
c=  uint8(42);  % car position
up=uint8(1);
right=uint8(2);
down=uint8(3);
left=uint8(4);
map_dim = 5;
map = map_in;
init_pos = current_pos;
pos = current_pos;
dir_selected = false;
t_dir_aux = current_dir;
map(init_pos) = t;
%================================================
%Check sorroundings
%================================================
up_value    =   map(pos-map_dim);
right_value =   map(pos+1);
down_value  =   map(pos+map_dim);
left_value  =   map(pos-1);
% I check if there is a preferred vertical movement possible, otherwise i
% check if there is a preferred horizontal movement possible
current_pos_row = floor((double(current_pos)-1)/5);
dest_row = floor((double(destination)-1)/5);
v_dist = dest_row - current_pos_row;
dist = double(destination) - double(current_pos);
% pref_dir_selection
if dest_row < current_pos_row
    dir_pref = up;
elseif dest_row > current_pos_row
    dir_pref = down;
elseif destination > current_pos
    dir_pref = right;
elseif destination < current_pos
    dir_pref = left;
else
    dir_pref = up;
    dir_selected = true;        % Destination reached
end
% New pref selection
switch dir_pref
```


Leonardo Ligabue
Università di Pisa

```
    case up
        if ((up_value == p || up_value == d) && ~dir_selected)
            pos = pos - map_dim;
            t_dir_aux = up;
            dir_selected = true;
        end
    case right
        if ((right_value == p || right_value == d) && ~dir_selected)
            pos = pos + 1;
            t_dir_aux = right;
            dir_selected = true;
        end
    case down
        if ((down_value == p || down_value == d) && ~dir_selected)
            pos = pos + map_dim;
            t_dir_aux = down;
            dir_selected = true;
        end
    case left
        if ((left_value == p || left_value == d) && ~dir_selected)
            pos = pos - 1;
            t_dir_aux = left;
            dir_selected = true;
        end
end
% New selection
if ((up_value == p || up_value == d) && ~dir_selected)
    pos = pos - map_dim;
    t_dir_aux = up;
    dir_selected = true;
end
if ((right_value == p || right_value == d) && ~dir_selected)
    pos = pos + 1;
    t_dir_aux = right;
    dir_selected = true;
end
if ((down_value == p || down_value == d) && ~dir_selected)
    pos = pos + map_dim;
    t_dir_aux = down;
    dir_selected = true;
end
if ((left_value == p || left_value == d) && ~dir_selected)
    pos = pos - 1;
    t_dir_aux = left;
    dir_selected = true;
end
% t pref selection
switch dir_pref
    case up
        if (up_value == t && ~dir_selected)
            pos = pos - map_dim;
            t_dir_aux = up;
            dir_selected = true;
        end
    case right
        if (right_value == t && ~dir_selected)
            pos = pos + 1;
            t_dir_aux = right;
            dir_selected = true;
        end
    case down
```

```
        if (down_value == t && ~dir_selected)
            pos = pos + map_dim;
            t_dir_aux = down;
            dir_selected = true;
        end
    case left
        if (left_value == t && ~dir_selected)
            pos = pos - 1;
            t_dir_aux = left;
            dir_selected = true;
        end
end
% Threaded selection
if (up_value == t && ~dir_selected)
    pos = pos - map_dim;
    t_dir_aux = up;
    dir_selected = true;
end
if (right_value == t && ~dir_selected)
    pos = pos + 1;
    t_dir_aux = right;
    dir_selected = true;
end
if (down_value == t && ~dir_selected)
    pos = pos + map_dim;
    t_dir_aux = down;
    dir_selected = true;
end
if (left_value == t && ~dir_selected)
    pos = pos - 1;
    t_dir_aux = left;
    dir_selected = true;
end
% New means a "p" block
up_new    = 0;
right_new = 0;
down_new  = 0;
left_new  = 0;
% Valid means a "t" block
up_valid    = 0;
right_valid = 0;
down_valid  = 0;
left_valid  = 0;
if (up_value == p || up_value == d)
    up_new = 1;
elseif up_value == t
    up_valid = 1;
end
if (right_value == p || right_value == d)
    right_new = 1;
elseif right_value == t
    right_valid = 1;
end
if (down_value == p || down_value == d)
    down_new = 1;
elseif down_value == t
    down_valid = 1;
end
if (left_value == p || left_value == d)
    left_new = 1;
elseif left_value == t
```

```
    left_valid = 1;
end
up_wall     = (map(init_pos-5) == w | map(init_pos-5) == b);
right_wall  = (map(init_pos+1) == w | map(init_pos+1) == b);
down_wall   = (map(init_pos+5) == w | map(init_pos+5) == b);
left_wall   = (map(init_pos-1) == w | map(init_pos-1) == b);
% If at least 3 adiacents grid cells are "b" or "w" the current position
% must be flagged as "b"
if (left_wall && up_wall && right_wall)
    map(init_pos) = b;
end
if (up_wall && right_wall && down_wall)
    map(init_pos) = b;
end
if (right_wall && down_wall && left_wall)
    map(init_pos) = b;
end
if (down_wall && left_wall && up_wall)
    map(init_pos) = b;
end
map(pos) = c;
if dir_selected == true
    route_sel_done = true;
else
    route_sel_done = false;
end
target_pos = pos;
target_dir = t_dir_aux;
map_update = map;
```

4.5.4 Rotate chart

Viene lanciato il trigger per la funzione di selezione della rotazione. Successivamente, in funzione del valore del parametro “rot” vengono settati i valori nel driver attraverso I2C oltre al valore del compare del counter per la rotazione:

- Rot = 0 , Nessuna rotazione, il programma salta i passi della rotazione
- Rot = 1 , Sinistra 90° Inverto il moto delle ruote sinistre move_counter_value = 21
- Rot = 2 , Destra 90° Inverto il moto delle ruote destre move_counter_value = 21
- Rot = 3 , Destra 180° Inverto il moto delle ruote destre move_counter_value = 42

Nel caso di “rot” 1,2 o 3 viene poi inviato il messaggio I2C per iniziare il moto settando la MODE2 con uscita totempole:

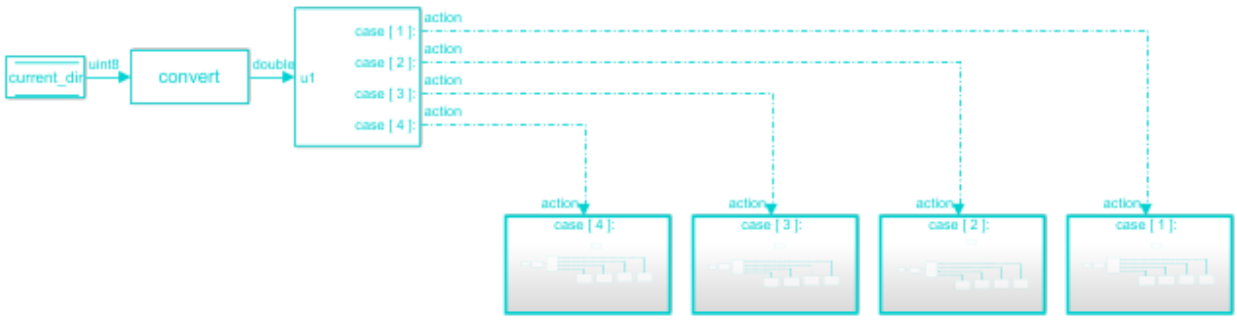
- Addr = 1
- Data = 7

Una volta raggiunto il counter di movimento specifico per ogni “rot” il flag “move_done” viene settato a 1 e quindi il programma prosegue fermando i motori attraverso il comando su MODE2:

- Addr = 1
- Data = 0

Successivamente al moto viene ripristinato il valore dei registri per il pilotaggio delle ruote in modo che ognuna di esse si muova in avanti. Infine, viene inviato il trigger per il clear dei flag di movimento.

La selezione della rotazione viene effettuata considerando i 16 possibili casi di combinazione fra direzione target e direzione corrente tramite l'utilizzo dei blocchi case di simulink:



4.5.5 Dir_update chart

Lancia il trigger della funzione dir_pos_update in cui avviene il controllo di posizione target e destinazione per verificare se dopo il movimento si raggiunge la destinazione e quindi settare il flag "stop" e inoltre aggiornare direzione e posizione correnti con i relativi valori target. Una volta effettuato l'assegnamento, l'update si considera concluso se posizioni e direzioni correnti e target sono uguali, settando quindi il flag "pos_update_done" e proseguendo allo stato successivo.

4.5.5.1 Codice dir_pos_update

```
function [current_pos, current_dir]  = fcn(target_pos, target_dir)
current_pos = target_pos;
current_dir = target_dir;
```

4.5.6 Move chart

Attiva il movimento e poi lo disattiva una volta raggiunto il counter impostato per il movimento con modalità analoghe a quelle usate negli step precedenti. Durante il passaggio allo step di stop del movimento, viene inviato il trigger "map_com" per la comunicazione UART della mappa. Infine, viene valutato lo stato del flag "stop":

- Stop = 0 → reitero il ciclo a partire dallo scan
- Stop = 1 → ho raggiunto la destinazione e quindi il veicolo si ferma nello stato destination_reached

4.5.7 Destination_reached chart

Stato finale che setta il flag "done" = 1 per la visualizzazione tramite LED del compimento del percorso.

5 Risultati

Durante lo sviluppo del progetto ogni singolo step è stato verificato con test funzionali. Il veicolo svolge correttamente le operazioni delle modalità 1 e 2 come visibile nei video allegati test_mode_1 e test_mode_2. Il sistema completo comunque necessita ancora di tuning e miglioramenti su diversi aspetti, come si evince dai video di test sul movimento ad esempio:

- Il servomotore perde momentaneamente la posizione durante la comunicazione I2C e non ne è stato ricercato il motivo.
- I 4 motori delle ruote non sono tarati con precisione e, soprattutto durante la rotazione, alcune delle ruote tendono a slittare, cosa dovuta anche all'uso di 4 ruote motrici al posto di 2 ruote motrici e di una ruota pirovante.
- La misura della distanza tramite LIDAR non è stata implementata, di conseguenza nemmeno il raffronto fra le 2 possibili mappe generate da SONAR e LIDAR è stato realizzato.
- Il cablaggio rende problematico l'utilizzo effettivo su una griglia reale, tant'è che i test delle modalità operative sono stati effettuati con il veicolo a ruote sollevate.

6 Allegati video

- **Rot_180_destra.mp4**
  Dimostrazione di rotazione a destra di 180°. Dal video si nota la problematica nella gestione della precisione di rotazione, tuttavia la rotazione, per quanto con margine di errore, avviene con successo.
- **Rot_90_sinistra.mp4**
  Dimostrazione di rotazione a sinistra di 90°. Dal video si nota la problematica nella gestione della precisione di rotazione, tuttavia la rotazione, per quanto con margine di errore, avviene con successo.
- **Movimento_50cm.mp4**
  Dimostrazione di movimento fra celle della griglia. Dal video si nota la problematica nella precisione con cui viene misurata la distanza, tuttavia lo spostamento, per quanto con margine di errore, avviene con successo
- **Simulazione modalità 1.mp4**
  Viene simulato il funzionamento in modalità 1 con quindi una mappa preimpostata. Dal video si può vedere il veicolo compiere le giuste azioni in riferimento alla mappa in termini di rotazione e movimento, inoltre la mappa viene correttamente aggiornata al passaggio del veicolo. Il sistema raggiunge la destinazione seguendo le modalità previste. Il log di evoluzione della mappa è il seguente:

wwwww wpwdw wpppw w*www wwwww	wwwww wpwdw w*ppw wbwww wwwww	wwwww w*wdw wtppw wbwww wwwww	wwwww wbwdw w*ppw wbwww wwwww	wwwww wbwdw wb*pw wbwww wwwww	wwwww wbwdw wbb*w wbwww wwwww	wwwww wbw*w wbbbw wbwww wwwww

- **Simulazione modalità 2.mp4**
  Viene simulato il comportamento in modalità 2 fornendo una mappa vuota al veicolo e utilizzando quaderni per simulare muri. Il veicolo rileva gli ostacoli e si muove di conseguenza nel percorso che viene generato spostando i quaderni opportunamente. (Il veicolo non è in grado con il codice attuale di seguire percorsi con ostacoli che cambiano posizione, una volta che un punto presenta un ostacolo,

per il veicolo quella cella della griglia sarà immutabilmente un muro). Il percorso seguito è conforme alla posizione degli ostacoli ed è il seguente:

ppppp pppdp ppppp p*ppp ppppp	ppppp pppdp p*ppp wtwpp ppppp	ppppp p*pdp wtwpp wtwpp ppppp	pwppp wt*dp wtwpp wtwpp ppppp	pwwpp wtt*p wtwpp wtwpp ppppp

- **Holder_hc-sr04.STEP & Holder_hc-sr04.STL**
  3D del supporto per il modulo SONAR
- **Sg90_holder.STEP & Sg90_holder.STL**
  3D del supporto per il servomotore