



SISTEMI DI GUIDA E NAVIGAZIONE

Integrazione di un laser Rplidar e sensori Pozyx per la localizzazione su una mappa preacquisita

Professore: Lorenzo Pollini

Paolo Cheli

Elisabetta Papallo

Benedetta Tarmati

Irene Valdambrini

INDICE



➤ SENSORI UTILIZZATI

- Rplidar A2/A3
- Sistema di riferimento Rplidar A2/A3
- Ultra wideband
- Sistema di riferimento uwb

➤ ACQUISIZIONE DELLA MAPPA

- Hector mapping
- Sistema di riferimento *map*
- Trasformata *frame_uwb/map*

➤ LOCALIZZAZIONE SULLA MAPPA

- Algoritmo di localizzazione Monte Carlo
- Algoritmo Mcl
- Pacchetto amcl
- amcl node
- Sistemi di riferimento richiesti da amcl
- Trasformata *odom/base_link*
- Coordinate di odometria tramite uwb
- Angolo di odometria tramite STM32
- Parametri amcl
- Nodo *initialpose.cpp*
- Funzionamento amcl su Rviz

➤ CONTROLLO MOTORI SU STM

- STM32F04
- Interfaccia Intel Joule e STM32
- Dati inviati dall'Intel Joule alla STM32
- Dati inviati all'Intel Joule dalla STM32
- STM32 - Debug Telemetry
- Controllo dei motori

➤ ESPERIMENTI

- Spostamento lungo y costanti
- Spostamento lungo x costanti
- Spostamento lungo una diagonale
- Raggiungimento waypoint

➤ INTERFACCIA GRAFICA

- File URDF
- File XACRO
- Simulazione in Rviz

➤ MOVEBASE

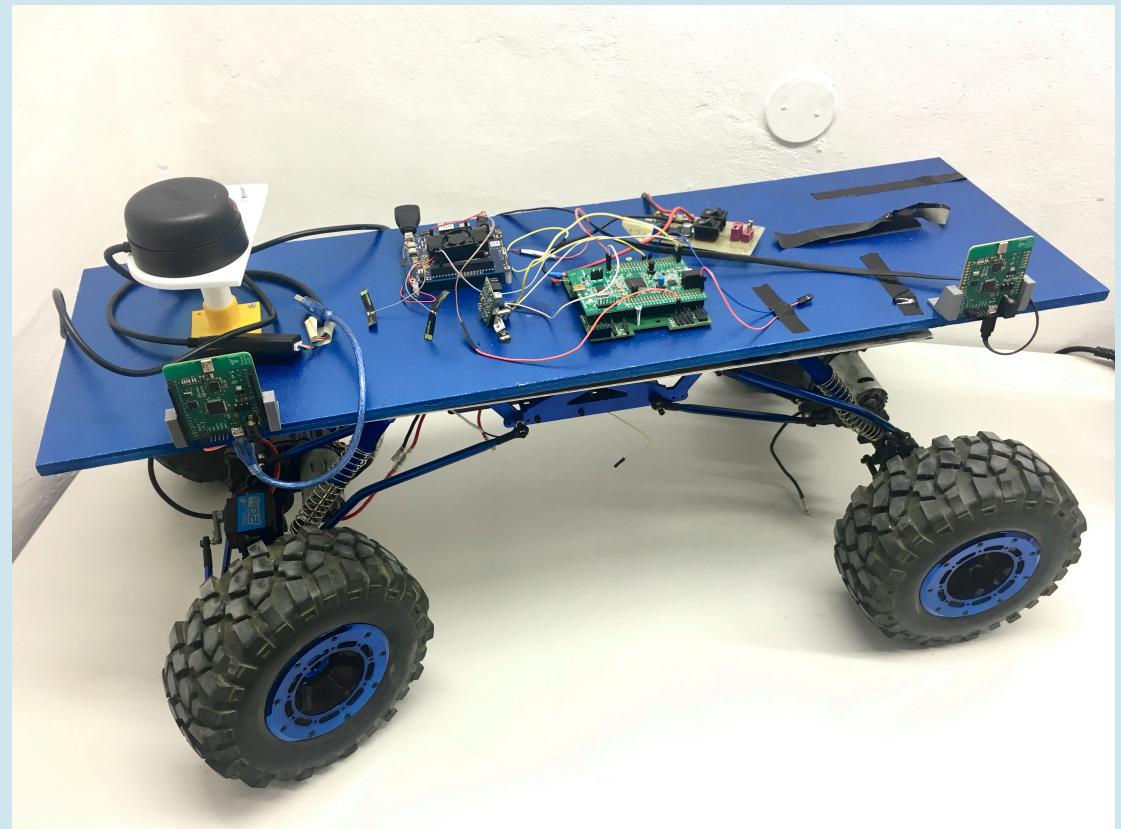
- movebase.cpp

➤ ACCORGIMENTI PER LAVORI FUTURI



SENSORI UTILIZZATI

- Laser 2D Rplidar A2/A3
- Ultra wideband (Pozyx)





• Rplidar A2

Caratteristiche:

- Laser 2D
- Range di scansione: 360°
- Frequenza di campionamento: 2000 - 8000 Hz
- Scan Rate: 5 - 15Hz
- Range distanze: 0.15 - 12m
- Baud rate: 115200 bps
- Risoluzione angolare: 0.45 - 1.35°





• Rplidar A3

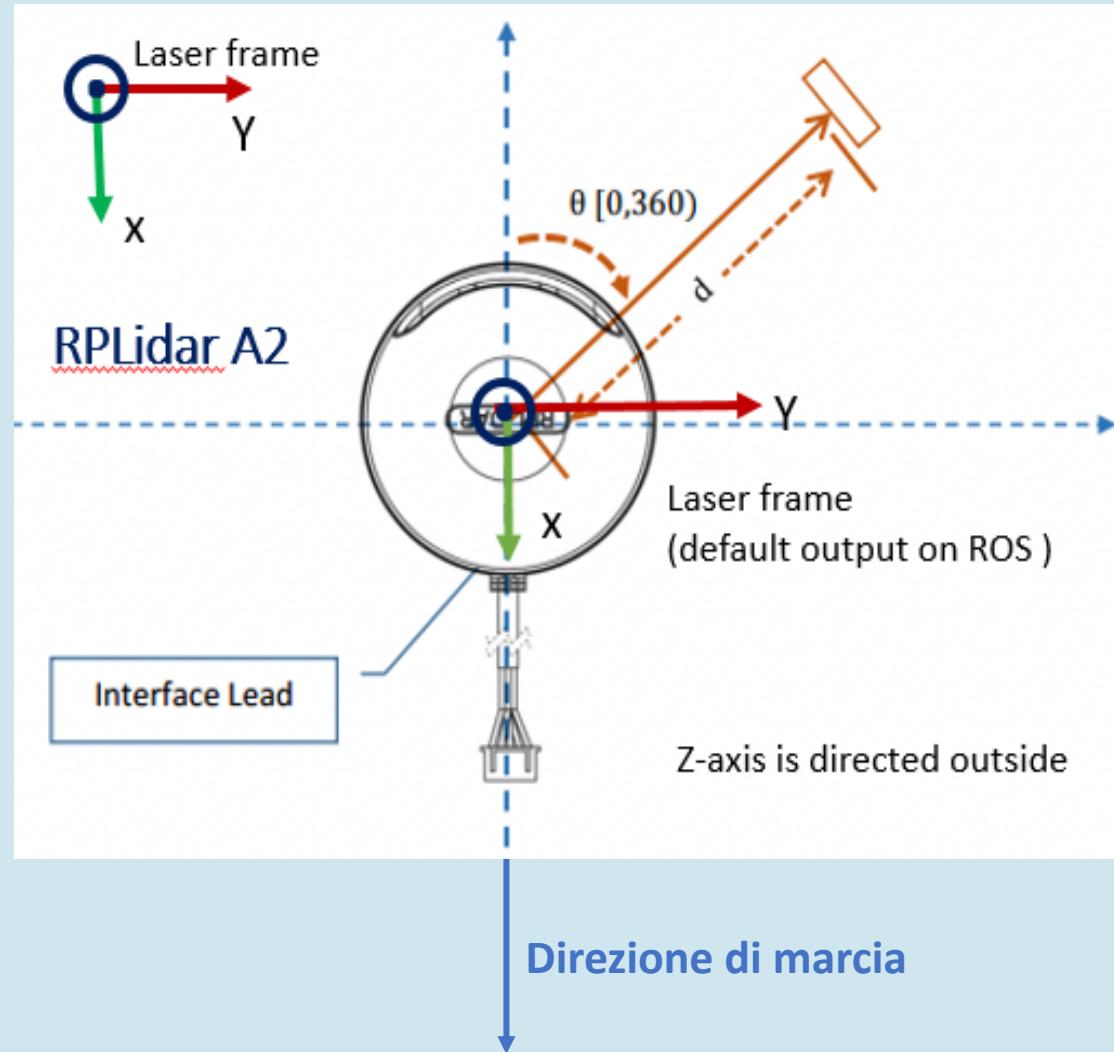
Caratteristiche:

- Laser 2D
- Range di scansione: 360°
- Frequenza di campionamento: 10000 - 16000 Hz
- Scan Rate: 10 - 20Hz
- Range distanze: 25 m
- Baud rate: 256000 bps
- Risoluzione angolare: 0.3375 – 0.54°





Sistema di riferimento Rplidar A2 / A3



Il sistema di riferimento del laser è una terna fissa rispetto alla macchina.

Questo risulta avere l'asse delle x rivolto verso la direzione di marcia della macchina, l'asse z verso l'alto e l'asse y di conseguenza a formare una terna levogira.

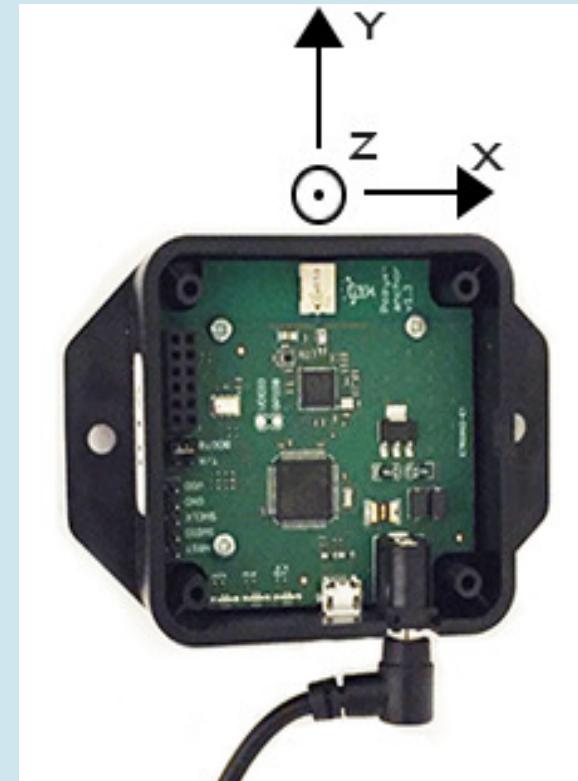
- Ultra wideband



Il sistema Pozyx viene utilizzato per il rilevamento della posizione di uno o più target tramite l'algoritmo di triangolazione.

Nel caso considerato il sistema è composto da sei elementi dotati di un'antenna uwb: 4 ancore e 2 tag.

Le ancore costituiscono il sistema di riferimento fisso (*frame_uwb*) rispetto al quale viene indicata la posizione delle tag, per questo è necessario posizionarle seguendo determinate convenzioni e mantenere invariata la loro collocazione a seguito della calibrazione.





Sistema di riferimento uwb

Le ancore UWB devono essere collocate con le seguenti convenzioni:

- l'ancora 0 (0x6902) è posta nell'origine del sistema di riferimento;
- l'ancora 1 (0x6e7a) definisce il verso positivo e la direzione dell'asse y;
- l'ancora 2 (0x6e44) definisce il verso positivo e la direzione dell'asse x;
- l'ancora 3 (0x6e6c) definisce il verso positivo dell'asse z e deve essere posta ad una quota superiore alle altre tre ancore, in modo da avere una z positiva verso l'alto.





ACQUISIZIONE DELLA MAPPA





Hector mapping

La mappa è stata acquisita tramite **hector_mapping**, nodo principale di Hector SLAM (Simultaneous Localization And Mapping) che si occupa di generare una mappa e di localizzare il robot all'interno di essa.

Hector_mapping per funzionare ha bisogno di ricevere dal laser la scansione 2D e dopo aver utilizzato dei filtri per eliminare i punti indesiderati, proietta i punti trovati nel frame laser sul piano.

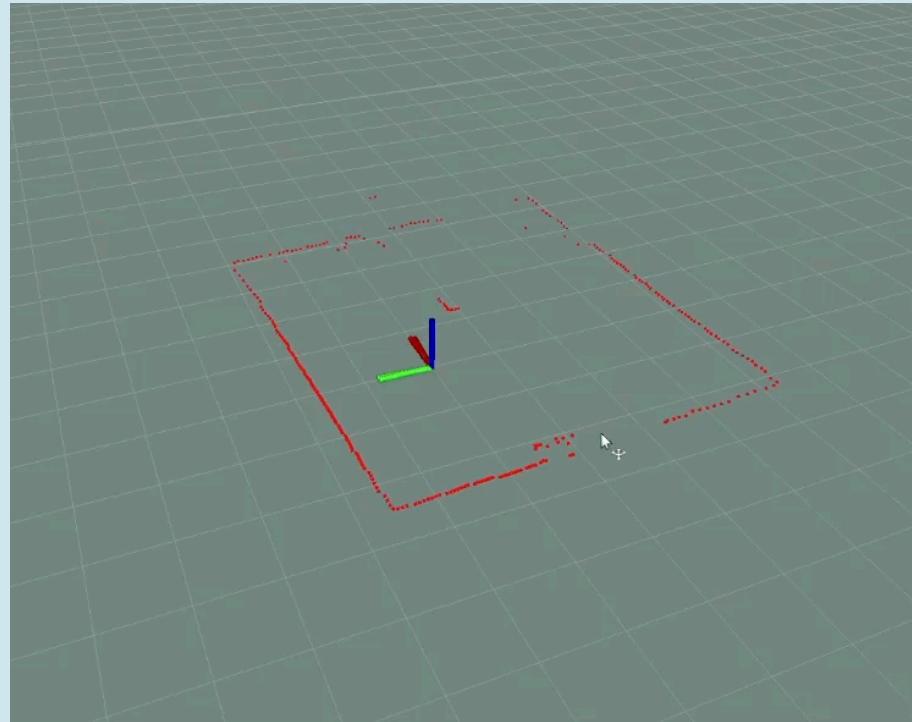
(Per ulteriori informazioni fare riferimento al progetto di Abbinante)

La localizzazione del robot avviene sulla mappa mentre questa viene costruita e non è stato possibile far localizzare il robot su una mappa preacquisita, questo perché la mappa caricata non ha alcun collegamento con la mappa registrata da **hector_mapping**, sulla quale avviene la localizzazione.



Sistema di riferimento *map*

La posizione e l'orientamento del laser all'inizio dell'acquisizione vanno a determinare la posizione e l'orientamento del frame *map*. La mappa viene salvata in formato .pgm e .yaml (formato con cui lavora amcl).

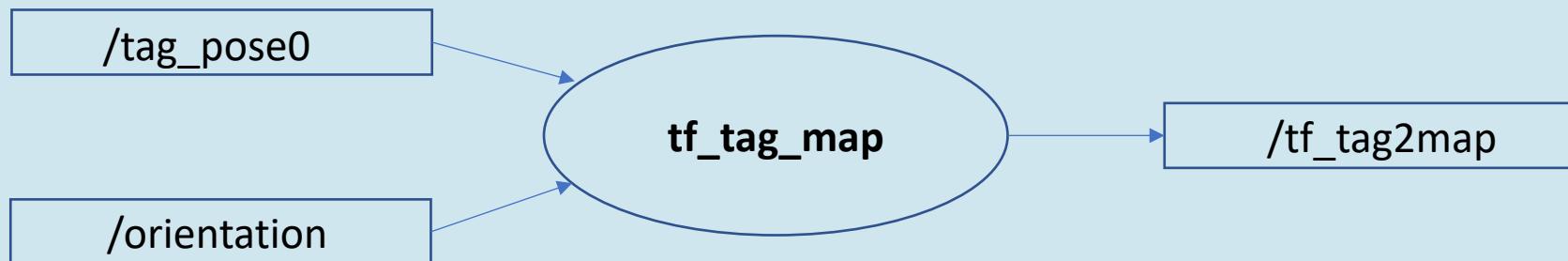




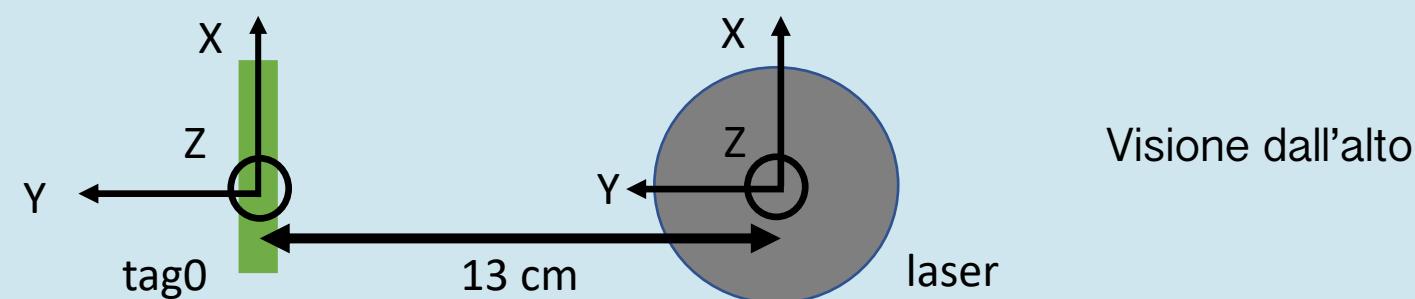
Trasformata *frame_uwb/map*

Avendo due sistemi di riferimento fissi, quello della mappa e quello delle uwb, è nata la necessità di calcolare una trasformata statica tra i frame *map* e *frame_uwb* in modo da poter esprimere le coordinate delle posizioni del robot nel sistema di riferimento della mappa.

È stato creato il nodo **tf_tag_map.cpp** che si sottoscrive ai topic **/tag_pose0** (posizione della tag zero nel sistema di riferimento uwb) e al topic **/orientation** (orientamento della macchina nel sistema di riferimento uwb ottenuto dal filtro AHRS) e pubblica sul topic **/tf_tag2map** la trasformata tra frame uwb e frame mappa.



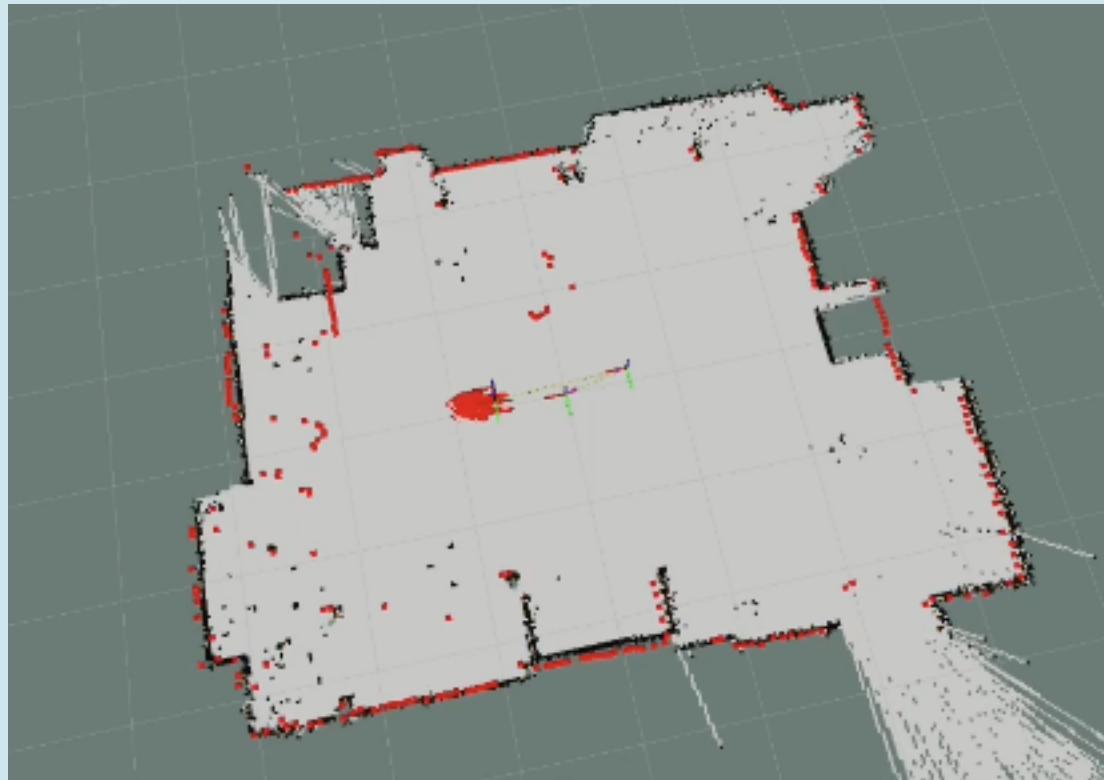
Questo nodo deve essere lanciato quando si acquisisce la mappa e stima la posizione del frame *map* rispetto al *frame_uwb* calcolando la media dei primi 50 valori della tag0. La posizione del laser non coincide con quella della uwb0 e quindi è stato aggiunto un fattore di correzione che tiene di conto della distanza (13 cm) tra la tag0 e il laser (fattore che andrà modificato nel caso in cui le posizioni dei sensori sulla macchina vengano cambiate).



Per i primi 5 secondi dell'acquisizione il laser deve rimanere fermo e può iniziare a muoversi solo quando sullo schermo compare la scritta "il robot può muoversi", questo perché alla fine dei 5 secondi viene fatta la media dei valori che sono stati registrati in questo intervallo di tempo, in cui la macchina non si è mossa, e i valori ottenuti andranno a determinare la posizione del frame *map*.



LOCALIZZAZIONE SULLA MAPPA





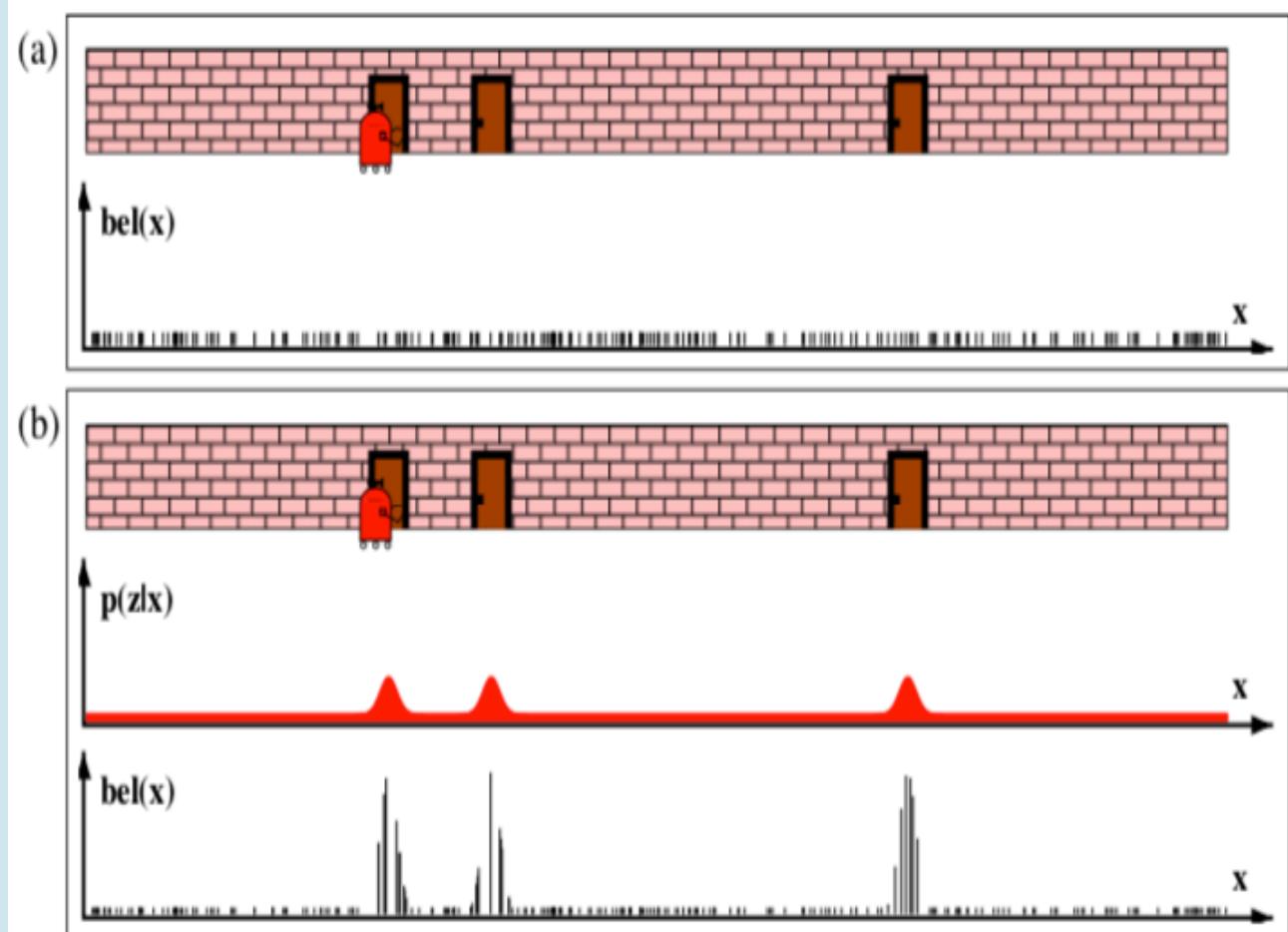
Utilizzando la trasformata tra *frame_uwb* e frame *map* è possibile localizzare il robot sulla mappa grazie ai dati ricevuti dalle uwb. Essendo le misure molto rumorose, per migliorare la localizzazione è stato utilizzato il nodo amcl che ha permesso l'integrazione dei valori delle uwb con i dati del laser.

Algoritmo di localizzazione Monte Carlo

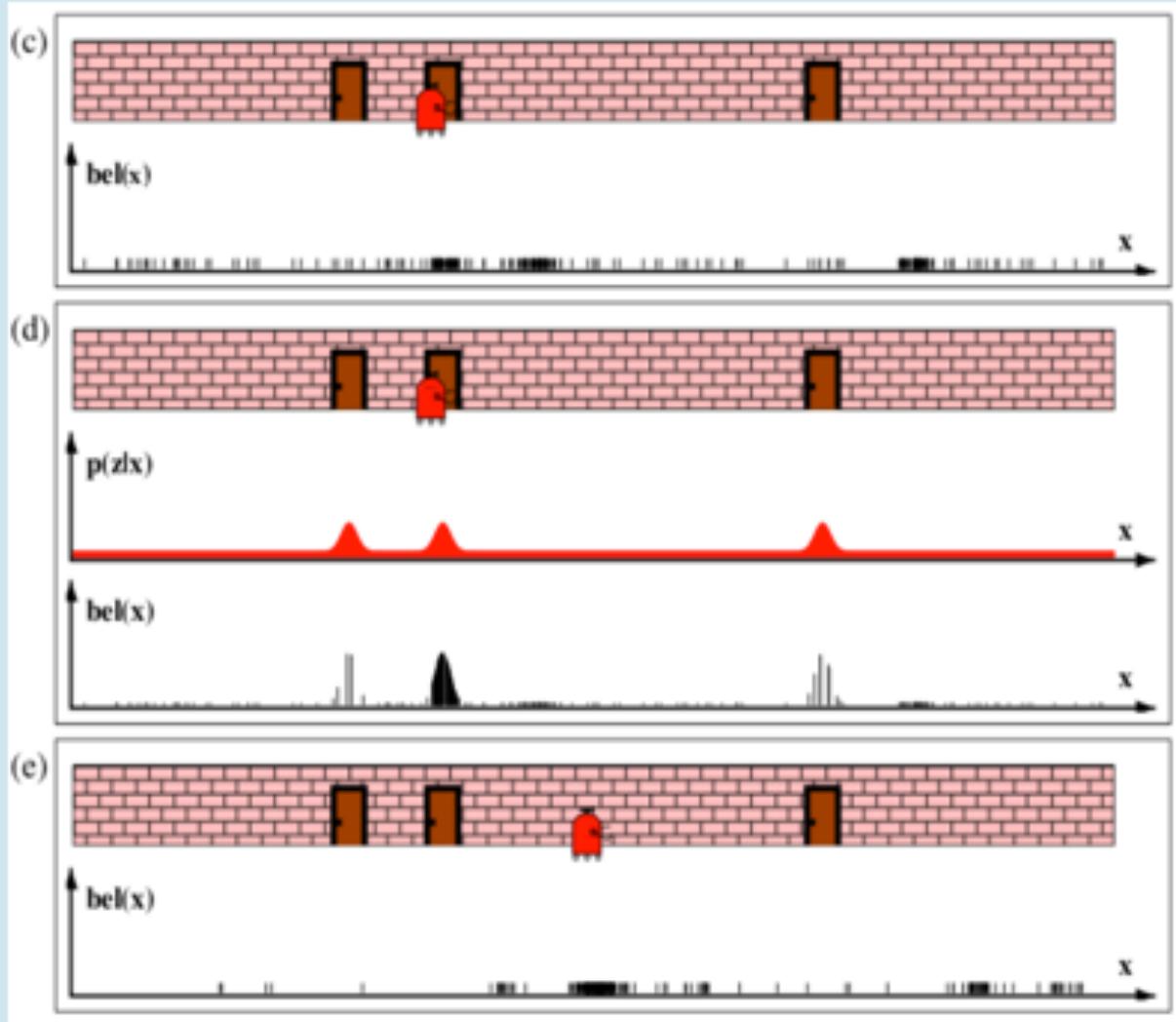
È un sistema di localizzazione che, data una mappa dell'ambiente, stima posizione e orientamento di un robot. L'algoritmo utilizza un filtro a particelle per rappresentare la distribuzione degli stati con probabilità più alta, in cui ogni particella rappresenta un possibile stato, in questo caso la posizione del robot nella mappa.

Inizialmente l'algoritmo distribuisce le particelle nello spazio di configurazione in modo random e uniforme, questo equivale a dire che la posizione del robot non è conosciuta. Quando il robot si muove, l'algoritmo predice la nuova distribuzione dei possibili stati e aggiorna le particelle grazie ai dati di un sensore laser. Dopo vari aggiornamenti le particelle convergeranno alla posizione reale del robot.

La figura mostra come funziona l'algoritmo di localizzazione nel caso di un robot che si muove lungo un corridoio. La figura (a) mostra le particelle distribuite in modo random e uniforme. In figura (b) è possibile vedere che appena il robot percepisce la porta, l'algoritmo assegna un peso ad ogni particella. Il set di particelle nella figura (b) è identico a quello in figura (a) e differisce solo per i pesi assegnati alle particelle. Un peso maggiore corrisponde ad una probabilità più alta di trovarsi in quello stato.



La figura (c) mostra il set di particelle dopo il ricampionamento, queste sono traslate perché viene tenuto di conto del movimento del robot. Tutte le particelle hanno lo stesso peso, ma sono più addensate nelle zone in cui è più probabile che sia il robot. Nella figura (d) viene assegnata una funzione peso alle nuove particelle e appare subito chiaro che la concentrazione più alta di particelle si ha in corrispondenza della seconda porta, questo significa che la posizione "robot davanti alla porta 2" è la più probabile. Successivi movimenti del robot porteranno a nuovi campionamenti.





Algoritmo MCL

Lo stato del robot dipende dall'applicazione considerata. Nel caso analizzato è descritto da $X=(x,y,\theta)$, dove x ed y sono le coordinate nel piano e θ l'orientamento del robot.

Nell'algoritmo di localizzazione Monte Carlo la stima dello stato è una densità di probabilità e all'istante t è rappresentata da M particelle $X_t = \{X_t^1, X_t^2, \dots, X_t^M\}$. L'algoritmo verifica la proprietà di Markov, per la quale la distribuzione delle probabilità dello stato corrente dipende solo dallo stato precedente.

All'istante t , l'algoritmo prende in input la distribuzione precedente $X_{t-1}=\{X_{t-1}^1, X_{t-1}^2, \dots, X_{t-1}^M\}$, il comando di attuazione u_t (proveniente dalle informazioni del sensore odometrico), i dati ricevuti dal laser z_t e restituisce in uscita la nuova distribuzione X_t .

Nel *motion_update*, l'algoritmo predice la nuova posizione del robot in base ai dati odometrici, aggiungendo a ciascuna particella dello stato precedente lo spostamento rilevato.

Nel *sensor_update* vengono aggiornati i pesi delle particelle in base a quello che il laser rileva nell'ambiente, infatti per ogni particella viene calcolata la probabilità che il robot, trovandosi in quello stato, possa vedere i dati registrati dal laser.

Il peso assegnato a ciascuna particella è proporzionale alla probabilità calcolata.

Algorithm MCL(X_{t-1}, u_t, z_t):

$$\bar{X}_t = X_t = \emptyset$$

for $m = 1$ to M :

$$x_t^{[m]} = \text{motion_update}(u_t, x_{t-1}^{[m]})$$

$$w_t^{[m]} = \text{sensor_update}(z_t, x_t^{[m]})$$

$$\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$$

endfor

for $m = 1$ to M :

draw $x_t^{[m]}$ from \bar{X}_t with probability $\propto w_t^{[m]}$

$$X_t = X_t + x_t^{[m]}$$

endfor

return X_t

Nella fase di ricampionamento la probabilità con cui l' algoritmo andrà a disegnare le particelle è proporzionale al loro peso.

Le particelle con peso più alto avranno maggiore probabilità di essere scelte.

In questo modo le particelle rappresenteranno una stima della posizione migliore di quella al passo precedente.

Algorithm MCL(X_{t-1}, u_t, z_t):

$$\bar{X}_t = X_t = \emptyset$$

for $m = 1$ to M :

$$x_t^{[m]} = \text{motion_update}(u_t, x_{t-1}^{[m]})$$

$$w_t^{[m]} = \text{sensor_update}(z_t, x_t^{[m]})$$

$$\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$$

endfor

for $m = 1$ to M :

draw $x_t^{[m]}$ from \bar{X}_t with probability $\propto w_t^{[m]}$

$$X_t = X_t + x_t^{[m]}$$

endfor

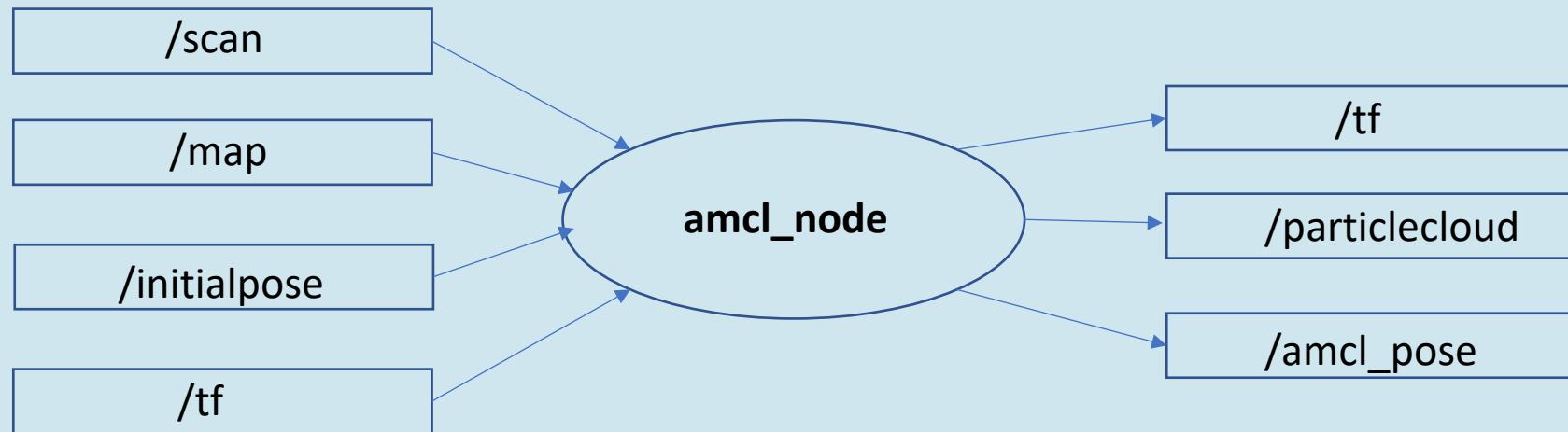
return X_t

Pacchetto AMCL

In ROS l'algoritmo di localizzazione viene implementato nel nodo **amcl_node.cpp** che si trova all'interno del pacchetto `navigation` (source: git <https://github.com/ros-planning/navigation.git>).

Topic a cui si sottoscrive:

- /scan
- /tf
- /initialpose
- /map



Topic su cui pubblica:

- /amcl_pose
- /tf
- /particlecloud

amcl_node



Topic a cui si sottoscrive:

- **/scan** : dati delle scansioni laser
- **/map** : informazioni della mappa
- **/tf** : amcl necessita delle trasformate *odom/base_link* e *base_link/laser*
- **/initialpose** : stima posa iniziale del robot nella mappa

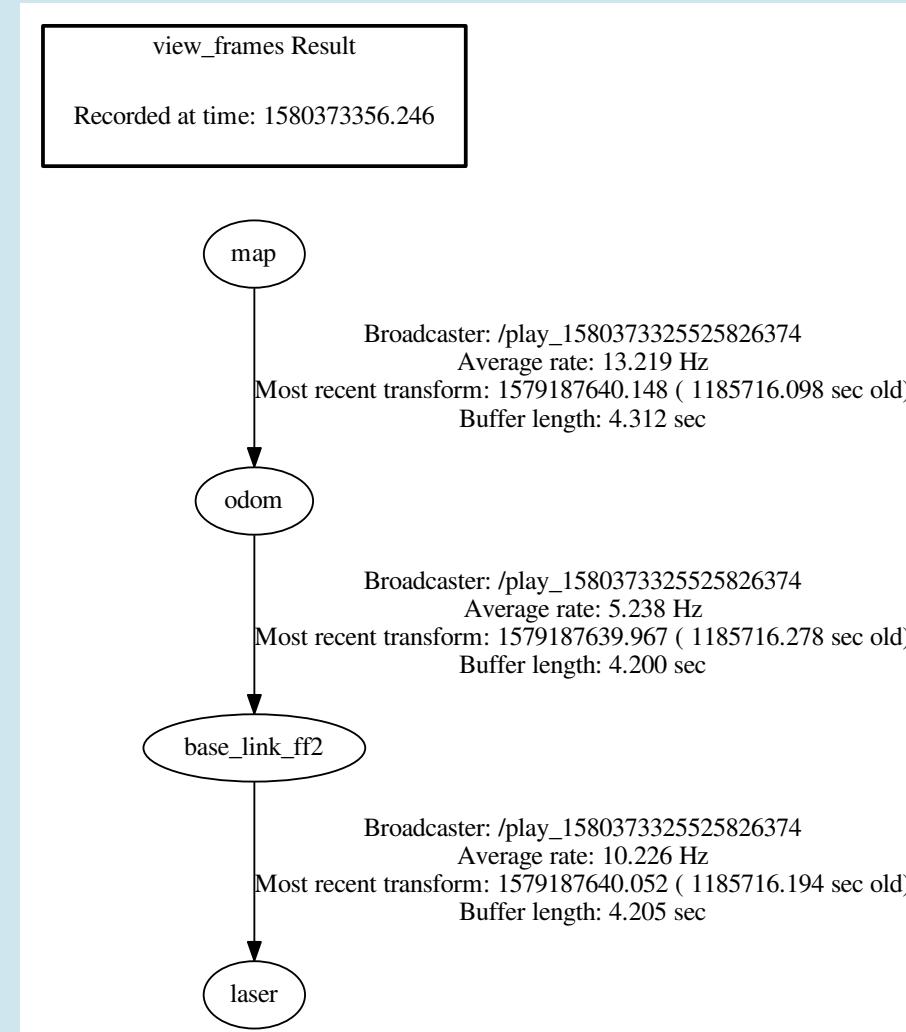
Topic su cui pubblica:

- **/amcl_pose** : stima della posa del robot fatta da amcl
- **/particlecloud** : particelle del filtro di amcl
- **/tf** : amcl pubblica la trasformata *map/odom*

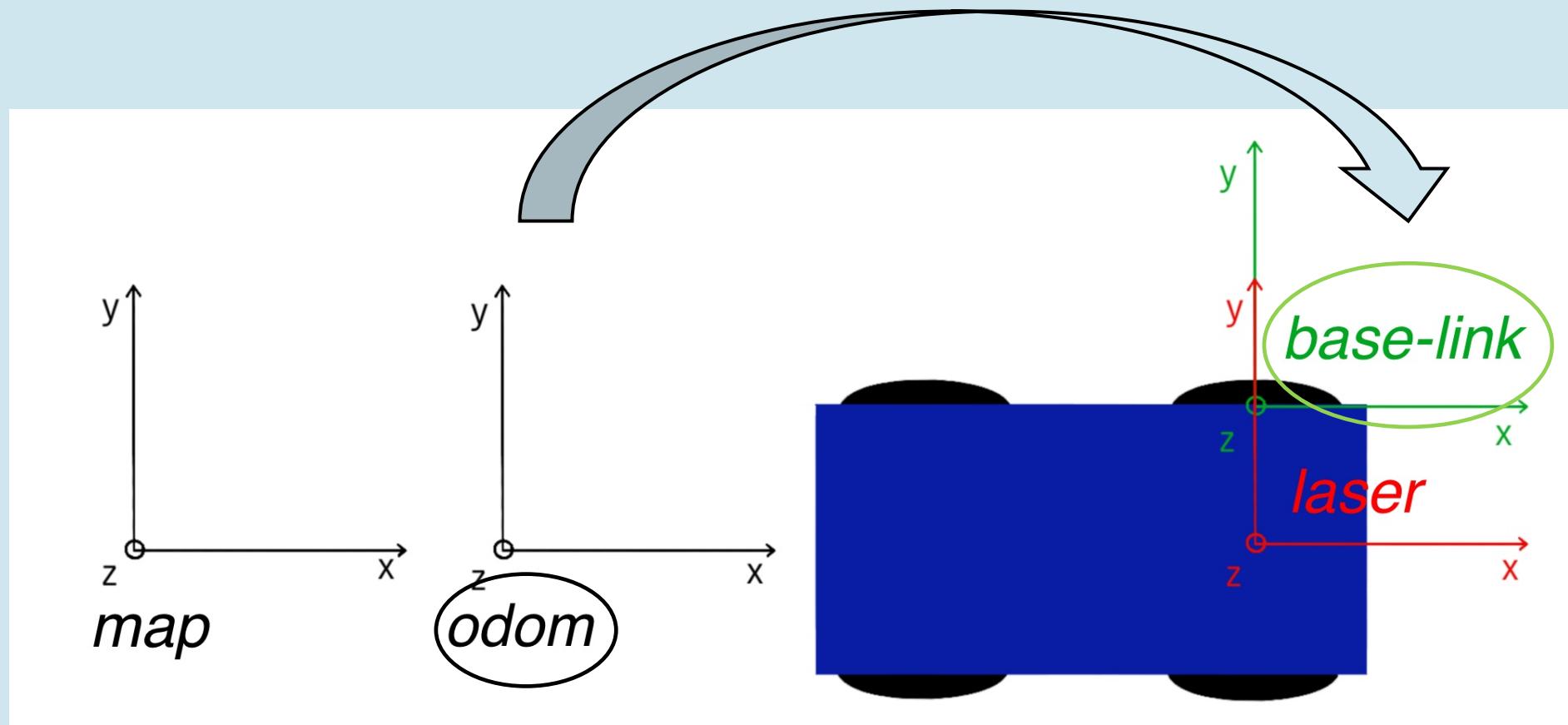


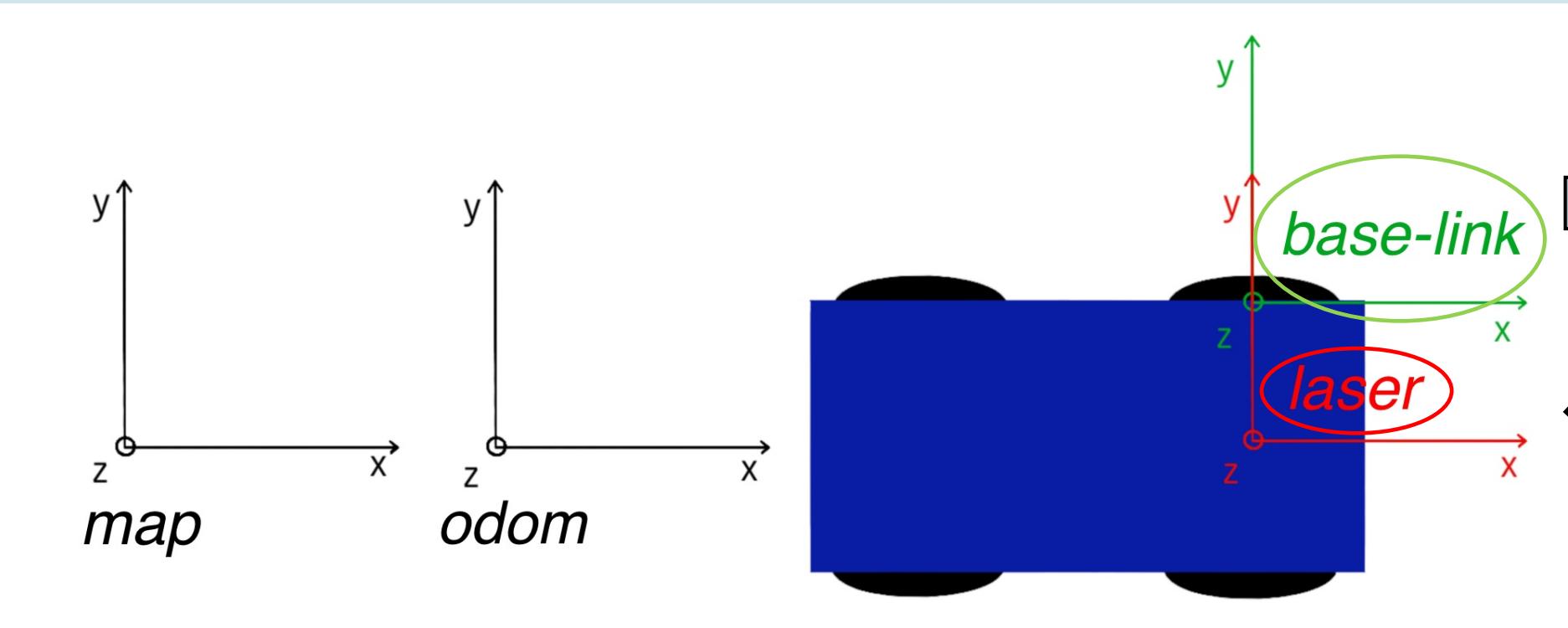
Sistemi di riferimento richiesti da amcl

- **map** : Sistema di riferimento fisso della mappa che ha origine nel punto in cui si inizia ad acquisirla
- **odom** : Sistema di riferimento che indica la posizione iniziale del robot
- **base_link** : Sistema di riferimento del robot posizionato sulla tag anteriore (tag0)
- **laser** : Sistema di riferimento del laser

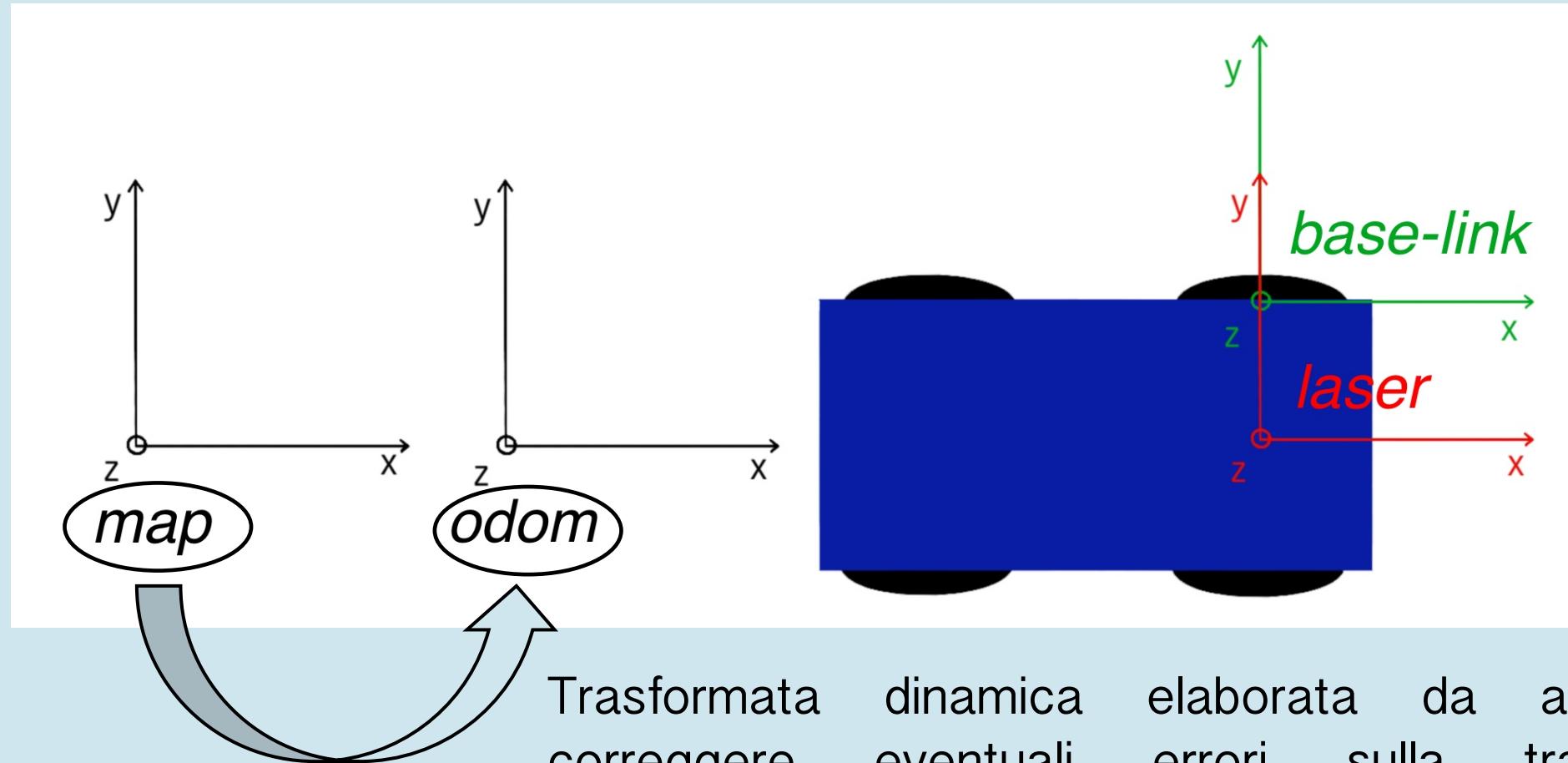


Trasformata che indica lo spostamento del robot rispetto alla sua posizione iniziale. Questa informazione è ricavata tramite i dati delle uwb.





Trasformata
statica specificata
nel launch di amcl
(*amcl_ff2.launch*)



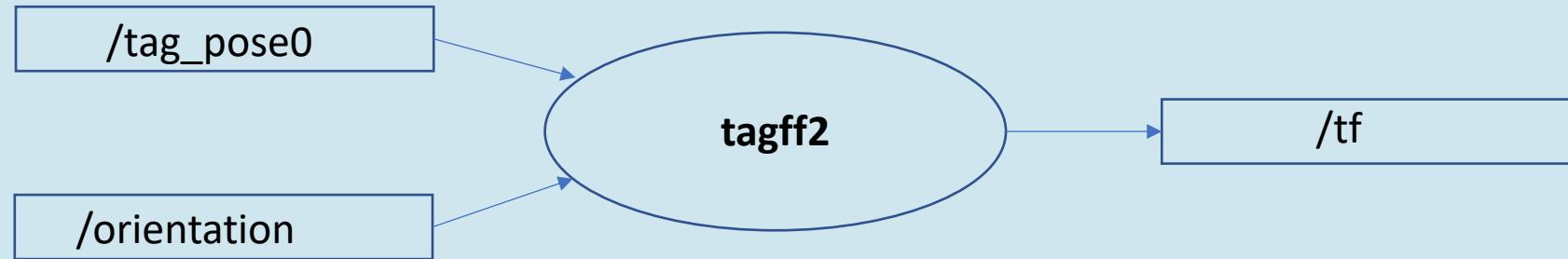
Trasformata dinamica elaborata da amcl per correggere eventuali errori sulla trasformata odom/base_link. Questo avviene sfruttando i dati del laser e permette una migliore localizzazione del robot nella mappa.



Trasformata *odom/base_link*

Il nodo amcl restituisce in uscita la trasformata *map/odom* e per calcolarla necessita di ricevere in ingresso due trasformate, *odom/base_link* e *base_link/laser*. Mentre *base_link/laser* è una trasformata fissa che dipende da come è stato montato il laser rispetto alla tag0 (*base_link*), *odom/base_link* è una trasformata che solitamente viene fornita da un sensore odometrico. In questo progetto l'informazione dell'odometria è stata data tramite i dati ricevuti delle tag.

Nel nodo **tagff2.cpp** vengono acquisite le coordinate della tag0 nel *frame_uwb*, queste successivamente vengono espresse nel frame *odom* e rappresentano lo spostamento del robot rispetto alla sua posizione iniziale. La stessa operazione è stata fatta con l'orientamento, prima espresso nel *frame_uwb*, poi riportato nel frame *odom*.





Coordinate di odometria tramite uwb

Essendo le misure delle uwb molto rumorose, è stato necessario filtrarle all'interno del nodo `tagff2.cpp` per migliorare le informazioni di odometria da fornire ad amcl.

Sono state valutate tre tipologie di filtri:

- **Filtro a media mobile** : media tra gli ultimi 5 campioni acquisiti dalle tag
- **Fading filter 1° ordine** : stima data da una combinazione lineare tra lo stato stimato al passo precedente e un guadagno che moltiplica il residuo.

$$\begin{cases} \hat{x}_n = \hat{x}_{n-1} + G[x_n^* - \hat{x}_{n-1}] = \beta \hat{x}_{n-1} + (1-\beta)x_n^* \\ G = 1 - \beta \end{cases}$$

- **Fading filter 2° ordine**

$$\begin{cases} \hat{x}_n = \hat{x}_{n-1} + \hat{x}_{n-1}T_s + G[x_n^* - (\hat{x}_{n-1} + \hat{x}_{n-1}T_s)] \\ \hat{x}_n = \hat{x}_{n-1} + \frac{H}{T_s}[x_n^* - (\hat{x}_{n-1} + \hat{x}_{n-1}T_s)] \\ G = 1 - \beta^2, H = (1 - \beta)^2 \end{cases}$$

Il guadagno β è associato alla lunghezza di memoria e varia tra 0 ed 1: aumentando il valore di β i campioni precedenti hanno una rilevanza maggiore nel calcolo della stima della posizione attuale.

\hat{x}_n : stima al passo n

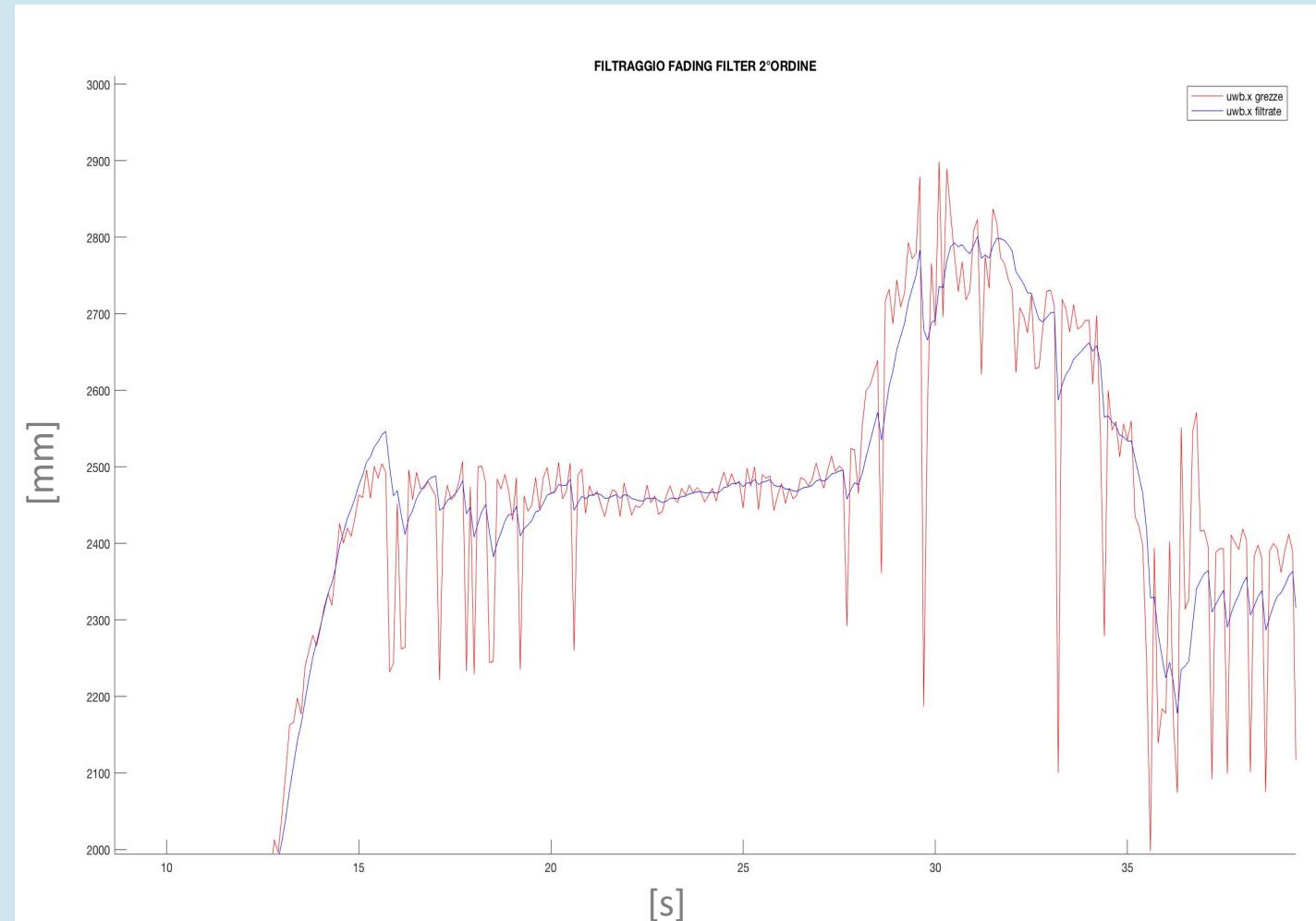
\hat{x}_{n-1} : stima al passo n-1

x_n^* : dati grezzi in ingresso al passo n



▪ Fading filter del 2° ordine

È stato scelto questo metodo di filtraggio ($\beta=0.9$) perché riesce a smussare i picchi del rumore delle uwb e non introduce un eccessivo ritardo.





Angolo di odometria tramite STM32

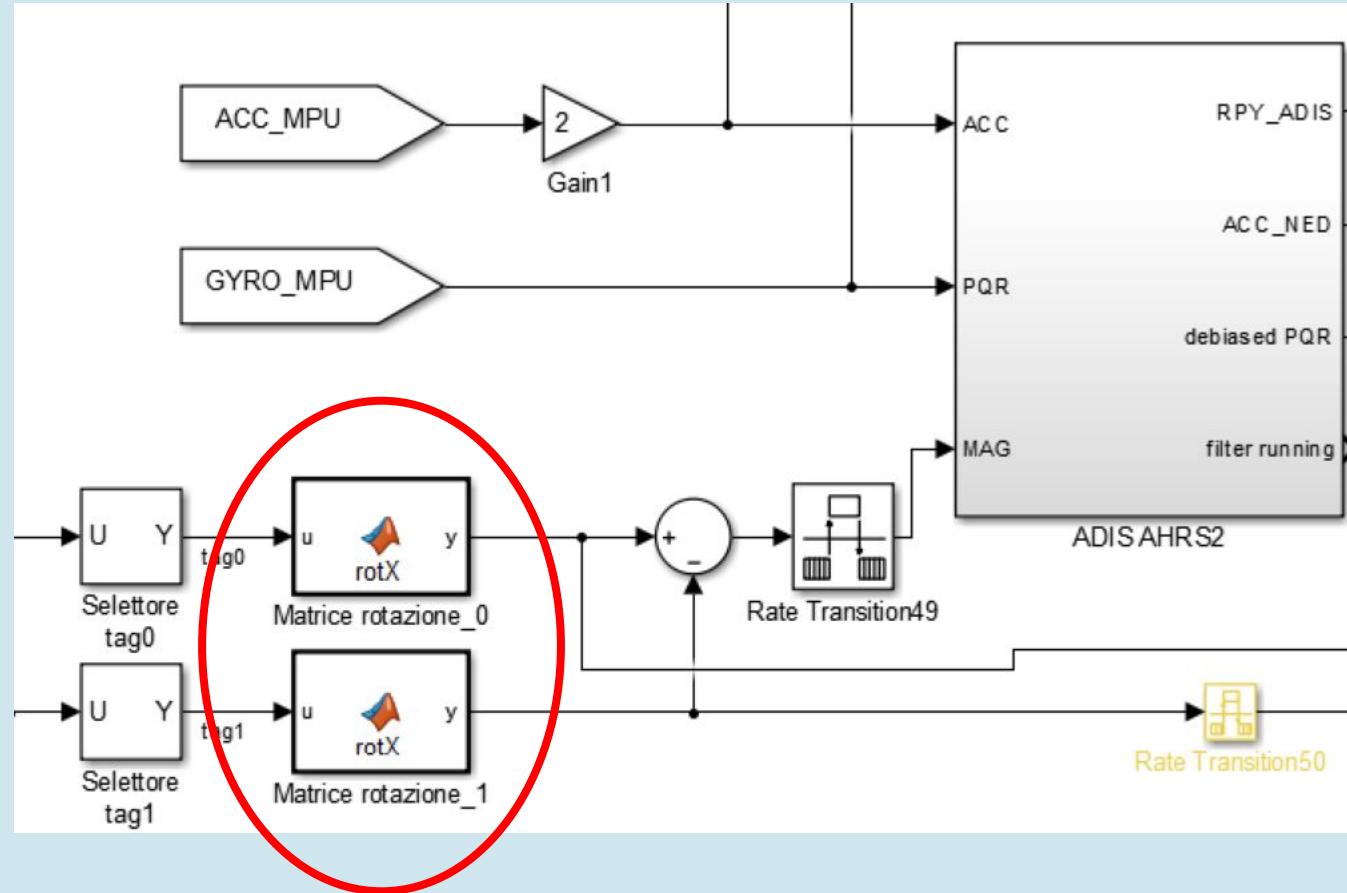
Per quanto riguarda l'orientamento dell'odometria, inizialmente era stato considerato l'angolo che il vettore congiungente le due tag formava con l'asse x positivo del frame *odom*. Data la rumorosità delle misure fornite dalle tag, l'angolo ottenuto aveva valori molto differenti tra loro in un breve arco temporale.

Per ovviare a questo problema è stato utilizzato l'orientamento stimato dal filtro AHRS2 nel firmware ICARO III presente sulla STM32.

Il filtro riceve in ingresso le misure date dalla IMU interna all'STM, ovvero dell'accelerometro e del giroscopio. In più riceve anche il vettore che collega le due tag e viene utilizzato dal filtro al posto del magnetometro.

A partire da questi tre ingressi il filtro restituisce in uscita accelerometro e giroscopio filtrati e gli angoli di Roll, Pitch e Yaw (nel *frame_uwb*).

L'angolo di Yaw, riportato nel sistema di riferimento *odom*, è stato utilizzato come stima dell'angolo di odometria.



Il filtro richiede in ingresso dei valori in un sistema di riferimento z-down, ma essendo il *frame_uwb* di tipo z-up, è stato necessario utilizzare una matrice R di rotazione intorno all'asse x.

L'ingresso u corrisponde ai valori x y z delle due tag.



Parametri amcl

Amcl necessita che vengano settati alcuni parametri che variano a seconda delle condizioni di utilizzo. Questi sono definiti all'interno del file amcl_ff2.launch.

PARAMETRO	VALORE	DESCRIZIONE
global_frame_id	map	Nome del frame globale
base_frame_id	base_link	Nome del frame del robot
odom_frame_id	odom	Indica il frame usato per l'odometria
odom_model_type	diff-corrected	Indica se il modello utilizzato ha le ruote omnidirezionali
resample_interval	1	Numero di aggiornamenti del filtro richiesti prima del ricampionamento delle particelle



Parametri amcl

PARAMETRO	VALORE	DESCRIZIONE
transform_tollerance	0.2	Ritardo sulla pubblicazione della trasformata (s)
min_particles	500	Numero minimo di particelle
max_particles	5000	Numero massimo di particelle
update_min_d	0.05	Movimento di traslazione richiesto prima di aggiornare il filtro (m)
update_min_a	0.1	Rotazione richiesta prima di aggiornare il filtro (rad)
initial_pose_x	0	Media sulle x della posa iniziale utilizzata per inizializzare il filtro (m)
initial_pose_y	0	Media sulle y della posa iniziale utilizzata per inizializzare il filtro (m)
initial_pose_a	0	Orientamento medio della posa iniziale utilizzata per inizializzare il filtro (rad)



Nodo *initialpose.cpp*

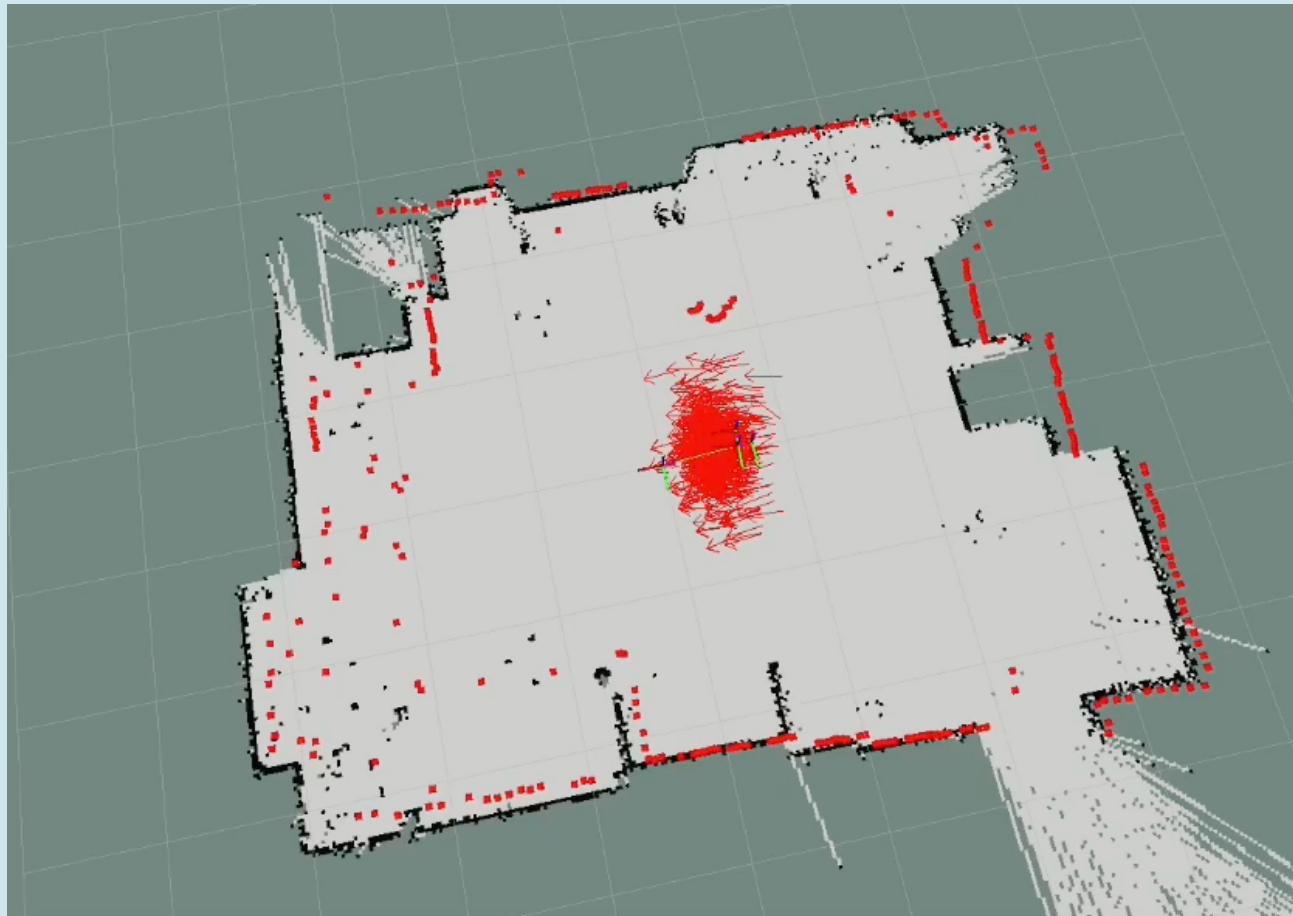
Una volta avviato amcl per inizializzare il filtro particellare è necessario conoscere una stima della posizione del robot nella mappa. È possibile fornire questa informazione tramite l'interfaccia grafica di Rviz con il comando "2D Pose Estimate" oppure lanciando il nodo **initialpose.cpp**.

Questo nodo si sottoscrive ai topic **/tag_pose0** e **/orientation** e pubblica sul topic **/initialpose** una stima della posizione iniziale del robot espressa nel frame *map*.



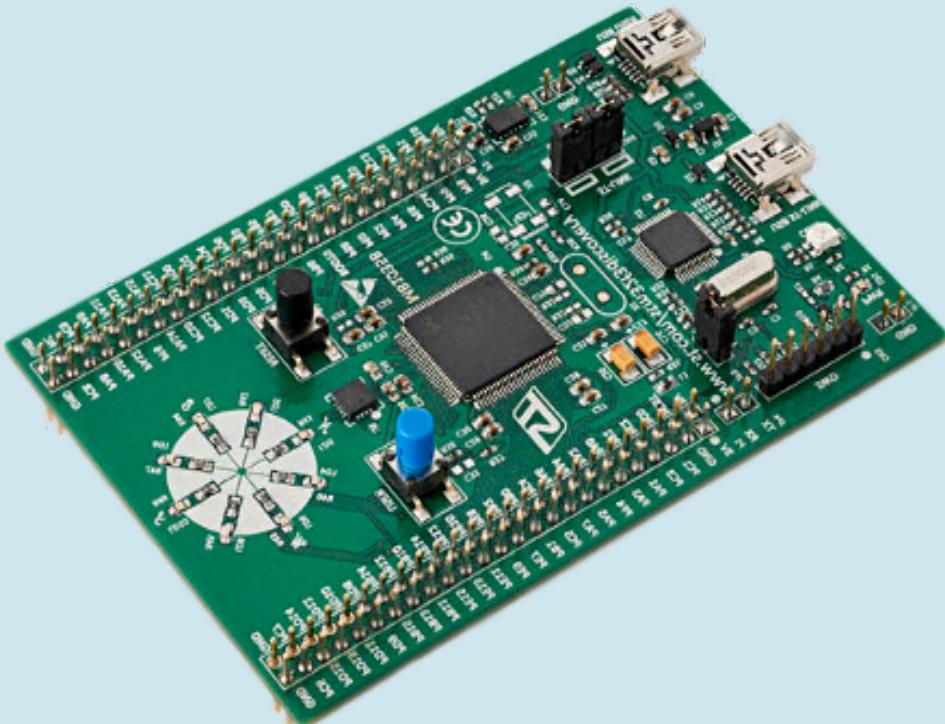


Funzionamento amcl su Rviz





CONTROLLO MOTORI SU STM

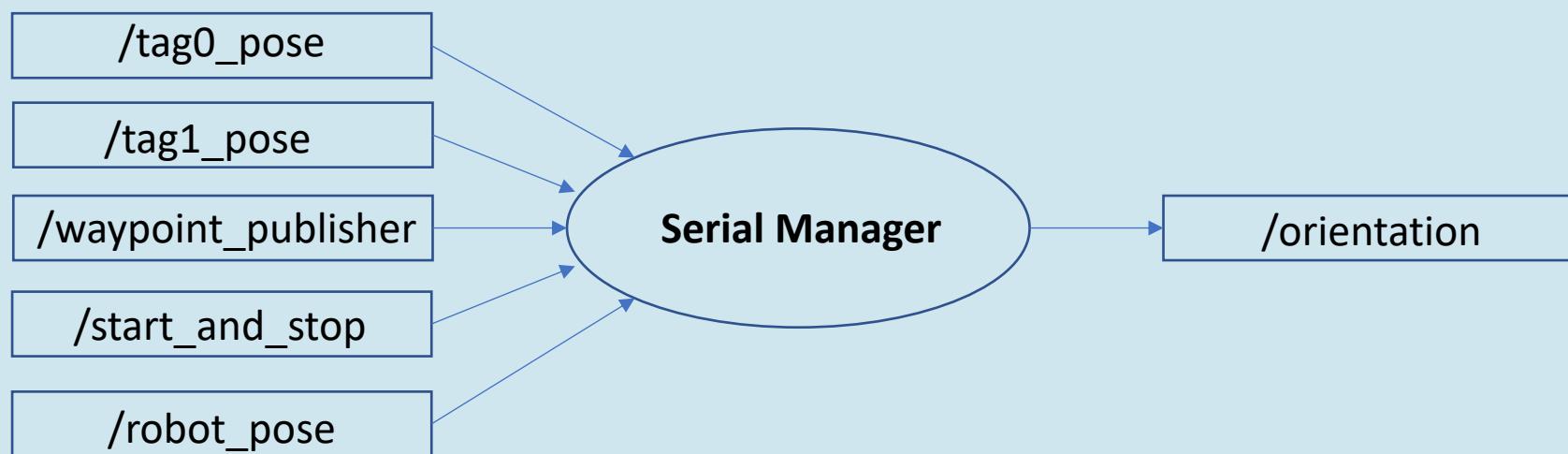




STM32F04

La scheda STM32 installata a bordo della macchina si occupa del controllo e dell'interfaccia con i sensori e con gli attuatori.

La comunicazione con l'Intel Joule, basata su protocollo seriale, avviene in scrittura ed in lettura tramite il nodo **Serial Manager**.





Interfaccia Intel Joule e STM

Per facilitare la comunicazione seriale fra Intel Joule ed STM sono stati implementati due nodi. La loro funzione è quella di ridurre il numero di variabili che devono essere inviate tramite Serial Manager, trasmettendo solo quelle strettamente necessarie. I nodi implementati sono:

- **amcl2robotpose**

Questo nodo legge dal topic */amcl_pose* e pubblica su */robot_pose* solo le coordinate e l'orientamento della macchina, trascurando le informazioni relative all'header e alla covarianza della stima della posa.

- **navgoal**

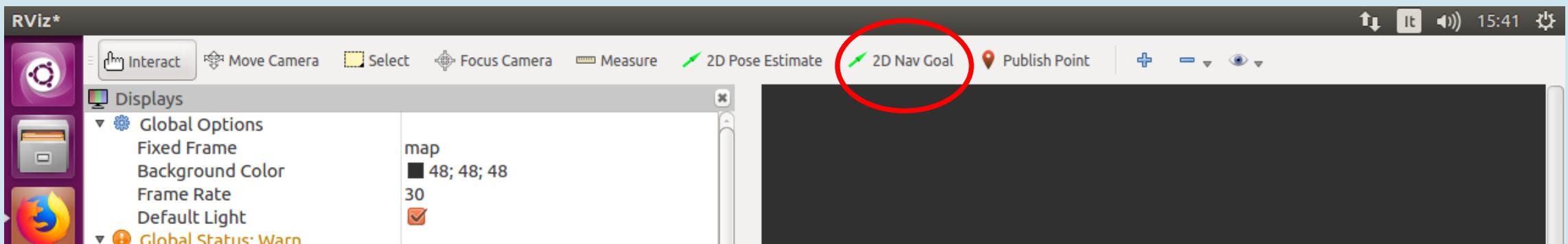
Questo nodo legge le coordinate (x,y) del waypoint dal topic */move_base_simple/goal* e le pubblica su */waypoint_publisher*, omettendo le informazioni relative all'header del messaggio.



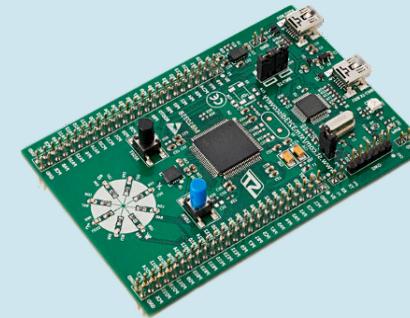
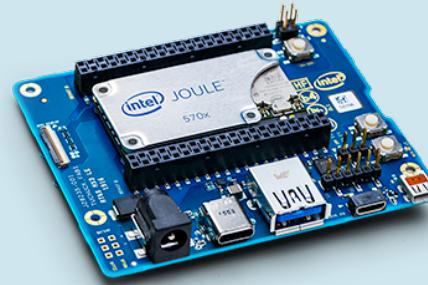
Per scegliere il waypoint è possibile selezionare il punto che si intende raggiungere tramite l'interfaccia grafica di Rviz con il comando “2D NavGoal”.

In questo modo, le coordinate del punto scelto vengono pubblicate sul topic **/move_base_simple/goal** ed il nodo **navgoal** le trasmette al topic **/waypoint_publisher**.

In alternativa, è possibile pubblicare direttamente le coordinate del punto sul topic **/waypoint_publisher**.



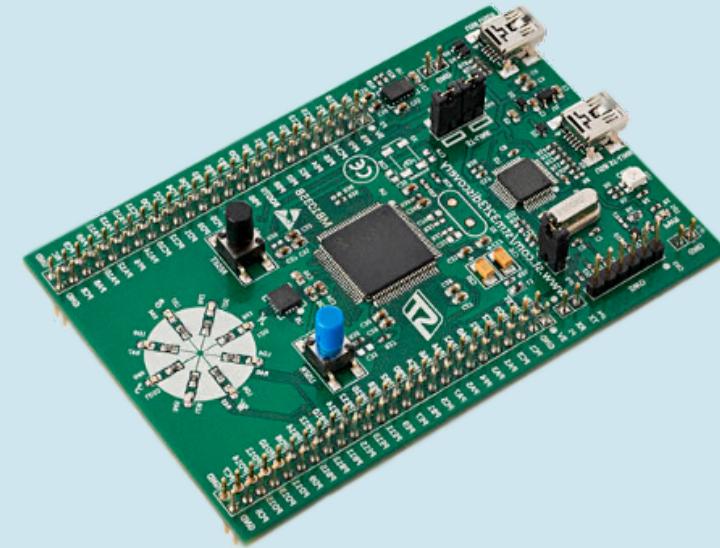
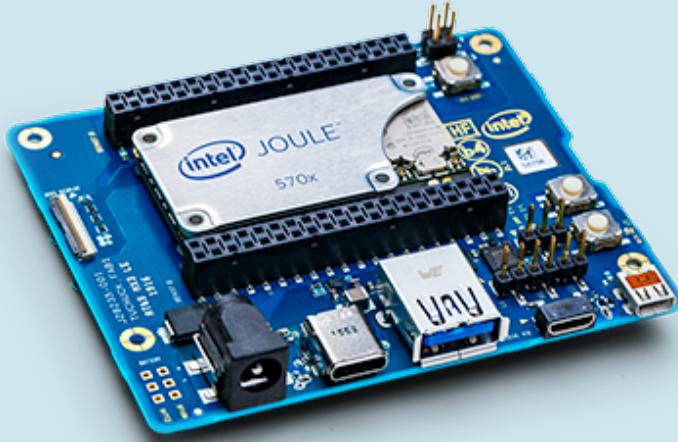
Dati inviati dall'Intel Joule alla STM32



- Posizione della tag0 e della tag1, tramite i topic ***/tag_pose0*** (Point) e ***/tag_pose1*** (Point)
- Posizione della macchina stimata da amcl e trasmessa tramite il topic ***/robot_pose*** (Point). Le prime due componenti corrispondono alla x e alla y della macchina nel sistema riferimento mappa, mentre la terza componente ne rappresenta l'orientamento.
- Posizione del waypoint che si vuole raggiungere, tramite il topic ***/waypoint_publisher*** (Point)
- Comando di start e stop per i motori, tramite il topic ***/start_and_stop*** (Float)

Per poter trasmettere l'informazione, la UART0TX dell'Intel Joule deve essere collegata con il pin U5 della STM32.

Dati inviati all'Intel Joule dalla STM32



- Angolo di orientamento della macchina

Per poter trasmettere le informazioni il pin U2 della STM32 deve essere collegato con la UART0RX dell'Intel Joule.

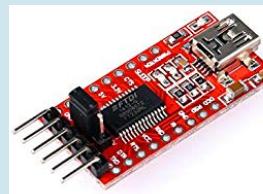
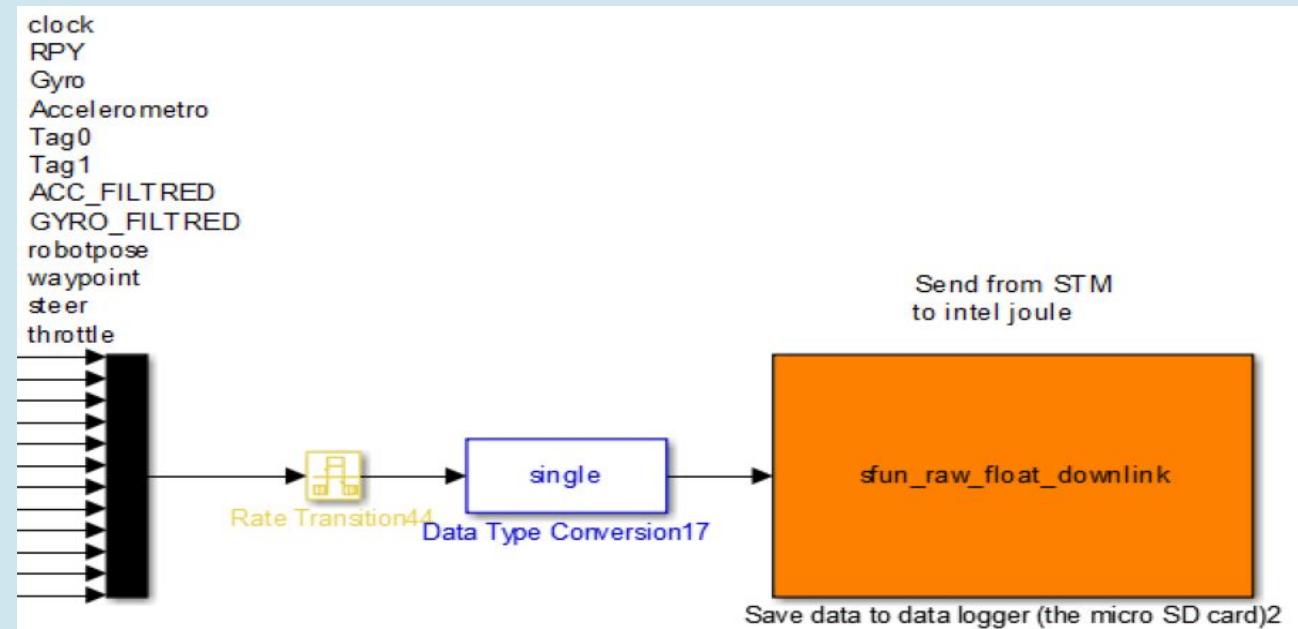


STM32 - Debug Telemetry

Tramite la telemetria è possibile trasmettere i dati di interesse in tempo reale.

Le variabili che si vogliono visualizzare devono essere mandate in ingresso alla `sfun_raw_flow_downlink`.

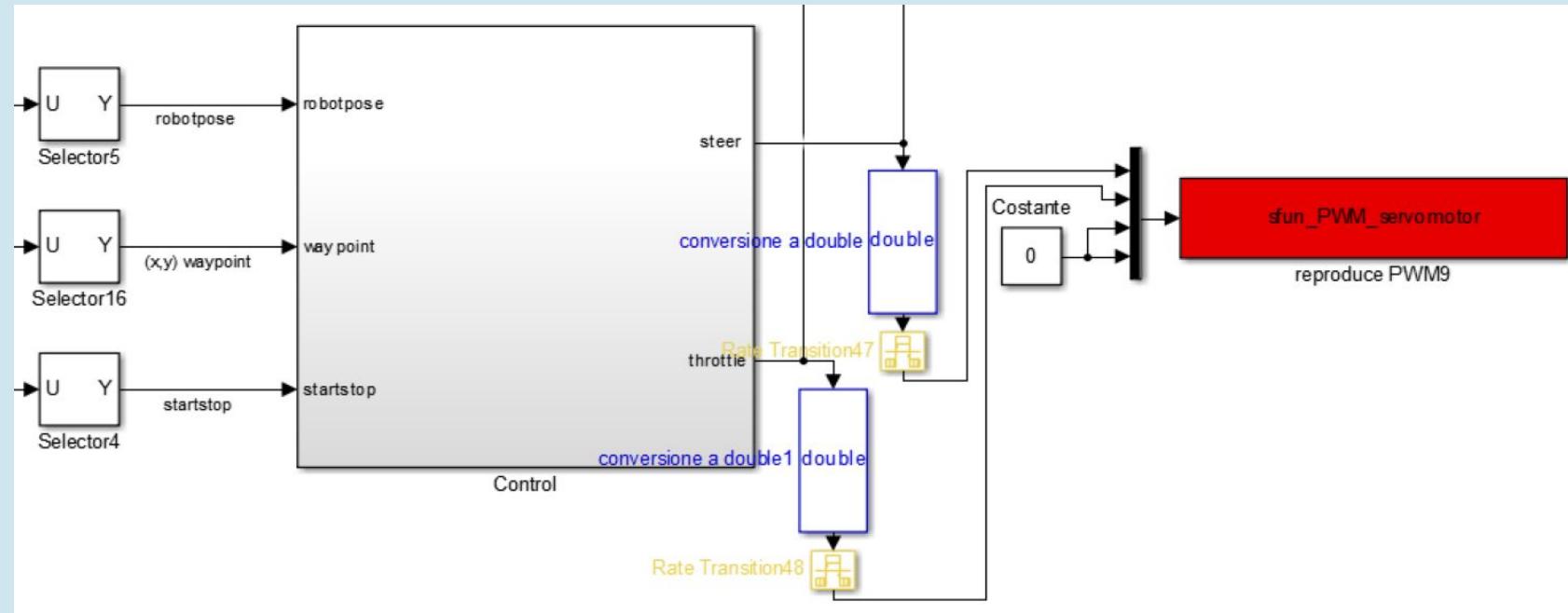
Per ricevere i dati in telemetria è necessario collegare il pin U3 della scheda STM32 al pin Rx dell'adattatore seriale.

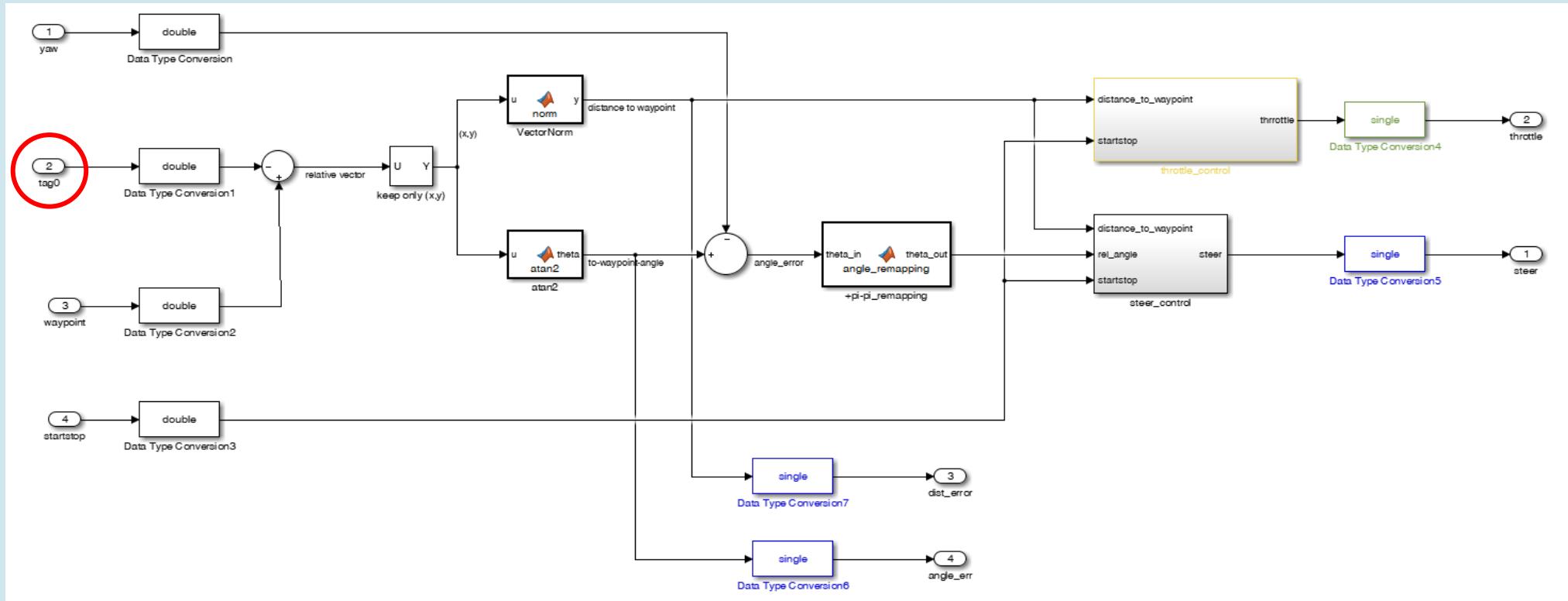




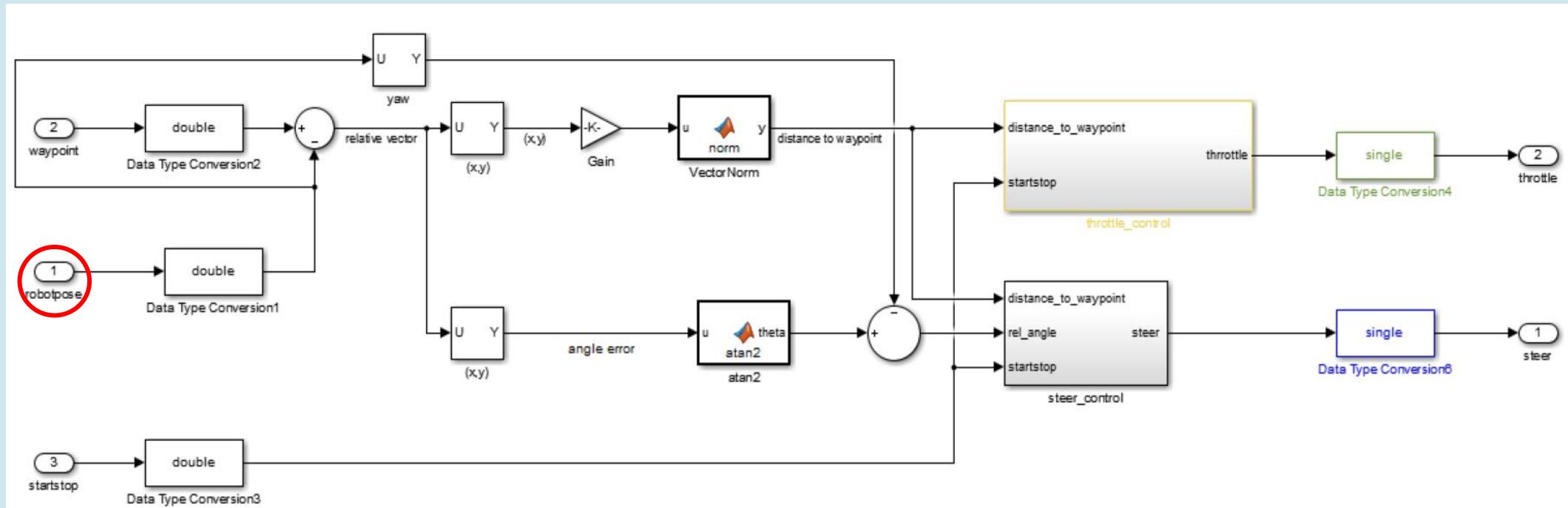
Controllo dei motori

Il controllo da mandare ai motori è stato implementato all'interno del blocco "Control". In ingresso riceve la posizione stimata da amcl (robotpose), le coordinate del goal (waypoint) e il comando di start e stop. Il controllore calcola i valori di pwm per lo steering e per il throttle, che permettono al robot di raggiungere il waypoint con un errore massimo di 25 cm.





Il controllo preesistente era di tipo proporzionale sia per il throttle che per lo steering. Per il throttle era stato implementato un controllore proporzionale alla distanza fra le coordinate (x,y) del waypoint e le coordinate (x,y) della tag0. Per lo steering il controllore era proporzionale all'angolo formato dalla direzione della macchina e il vettore congiungente la tag frontale con il waypoint.



Per il throttle è stato implementato un controllore proporzionale alla distanza fra le coordinate (x,y) del waypoint e le coordinate (x,y) della macchina stimata da amcl.

Per lo steering è stato utilizzato un controllore proporzionale all'angolo formato dalla direzione della macchina (stimata da amcl) e il vettore che la congiunge con il waypoint.



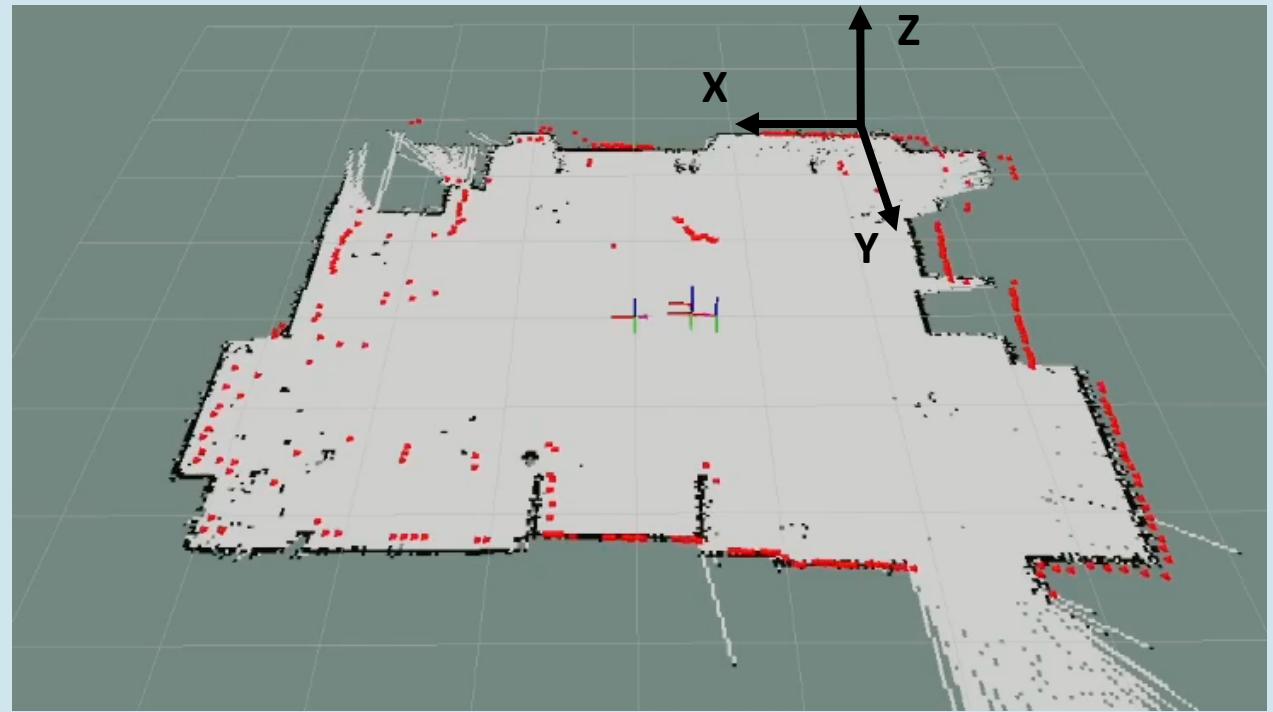
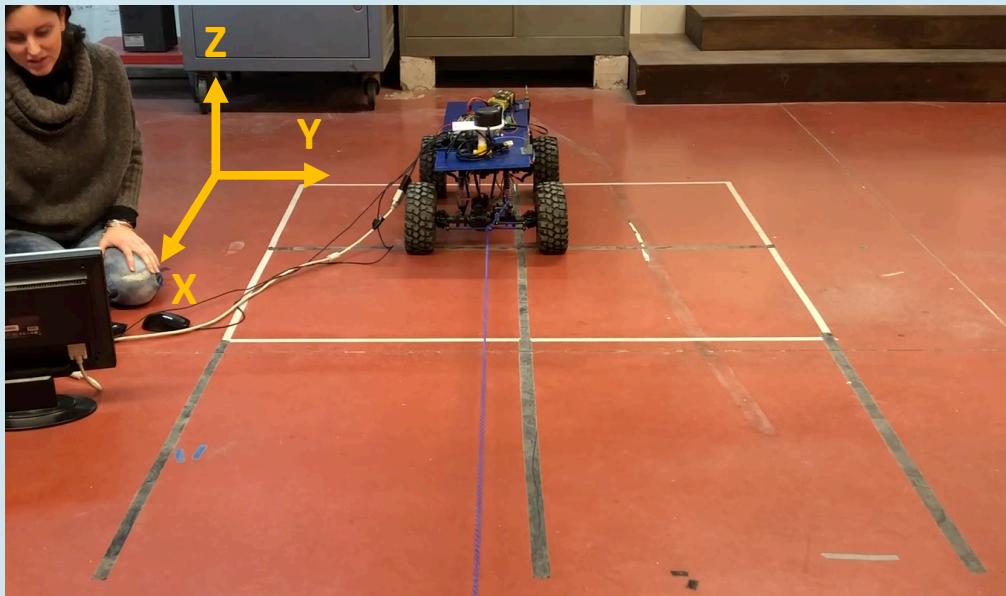
ANALISI DEI RISULTATI DI LOCALIZZAZIONE

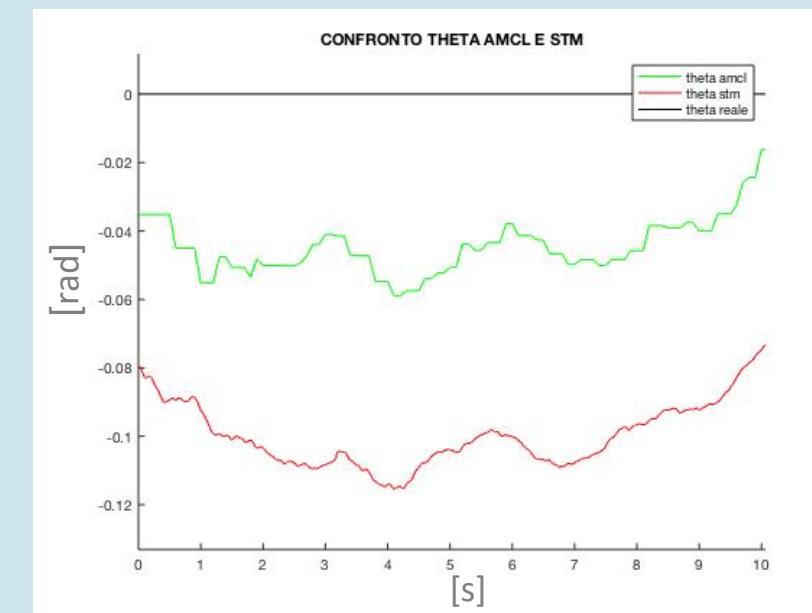
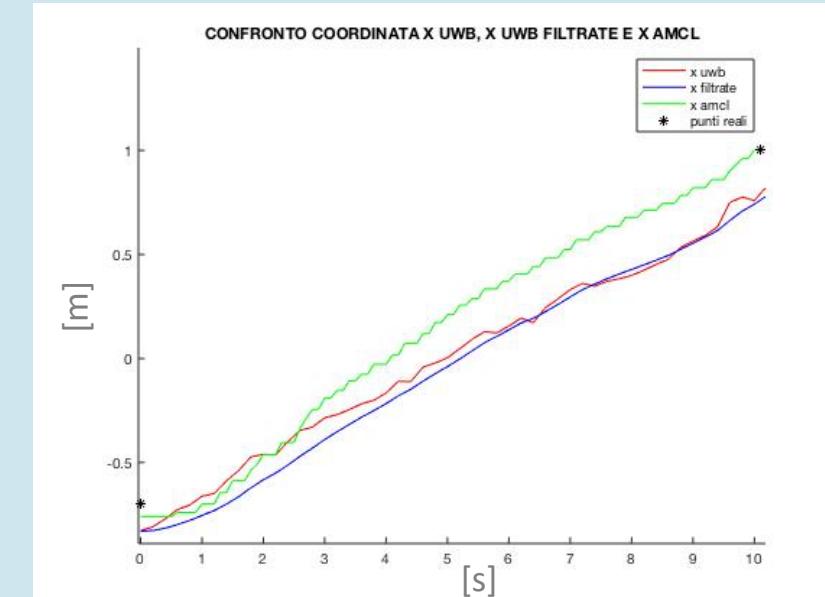
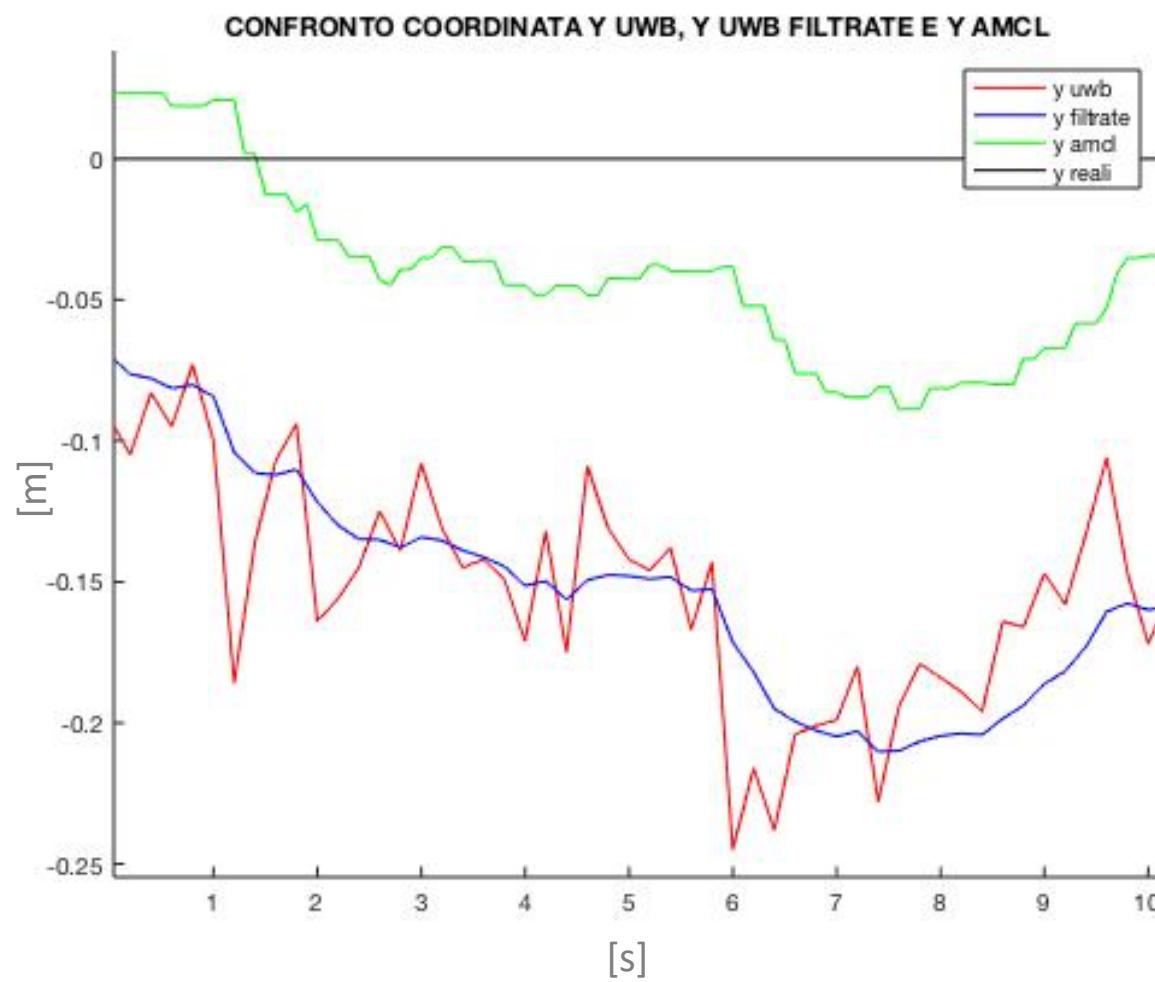
Per valutare i risultati ottenuti con l'algoritmo di localizzazione, sono stati condotti quattro tipi di esperimenti :

- Spostamenti con y costante
- Spostamenti con x costante
- Spostamenti in diagonale
- Raggiungimento waypoint

Spostamento lungo y costanti

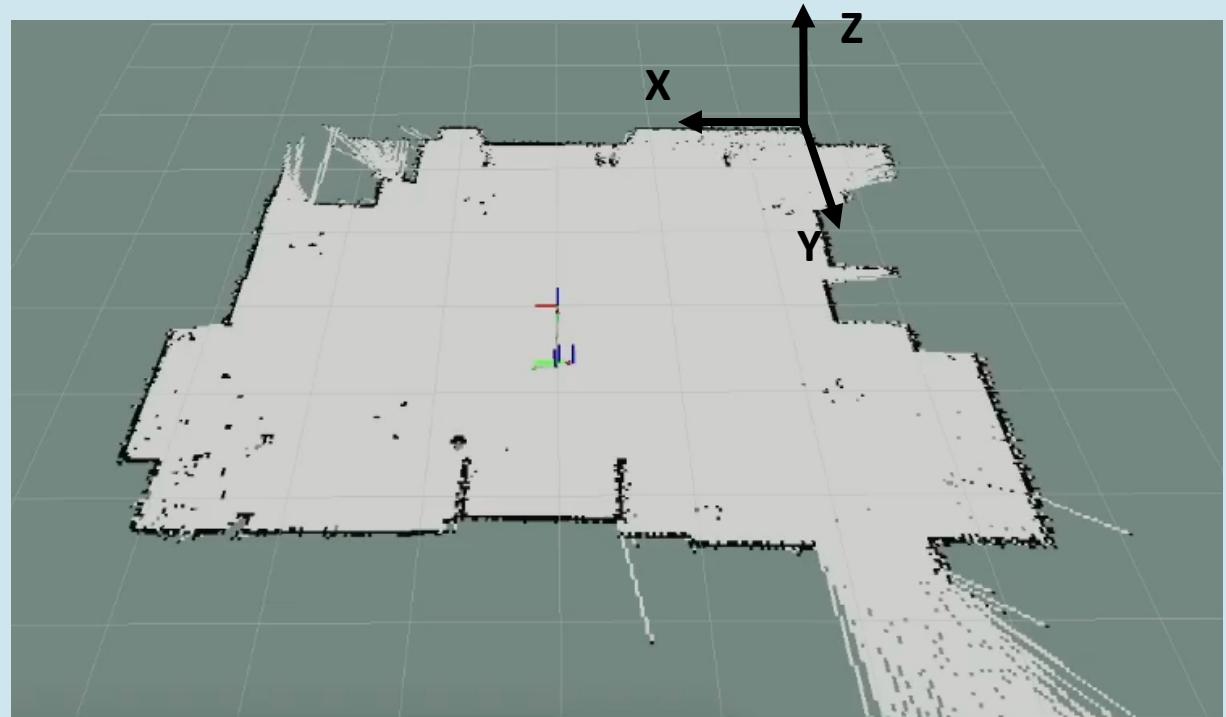
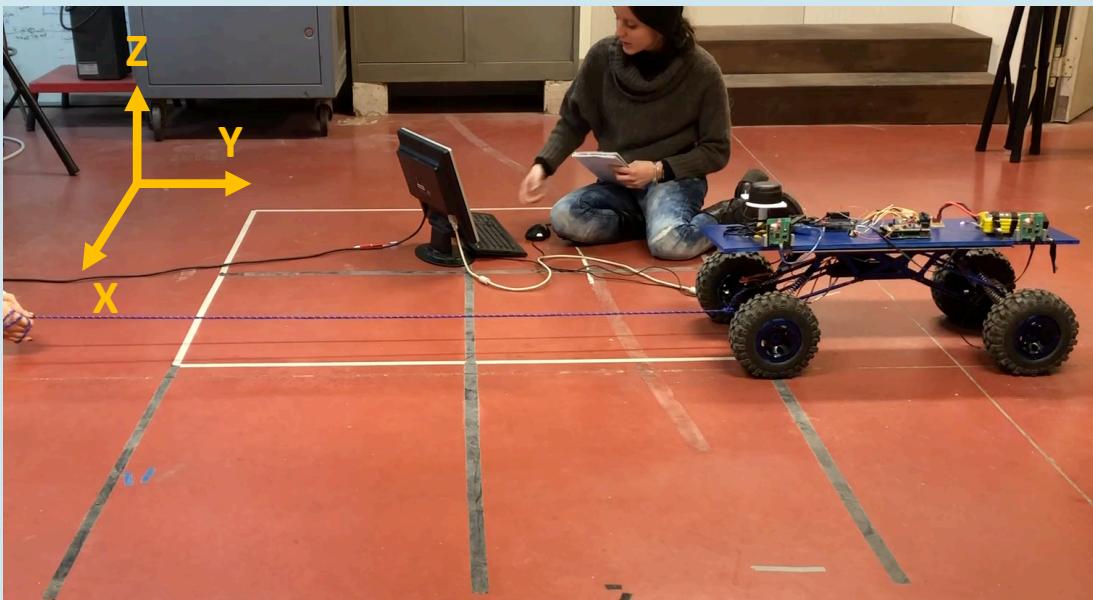
Il robot è stato fatto muovere con y costante pari a 0 m (nel frame *map*).

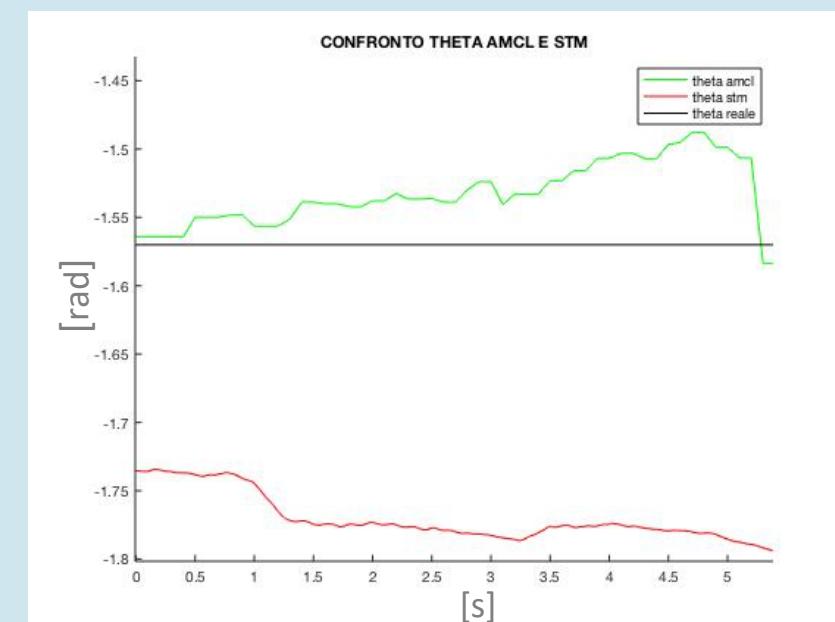
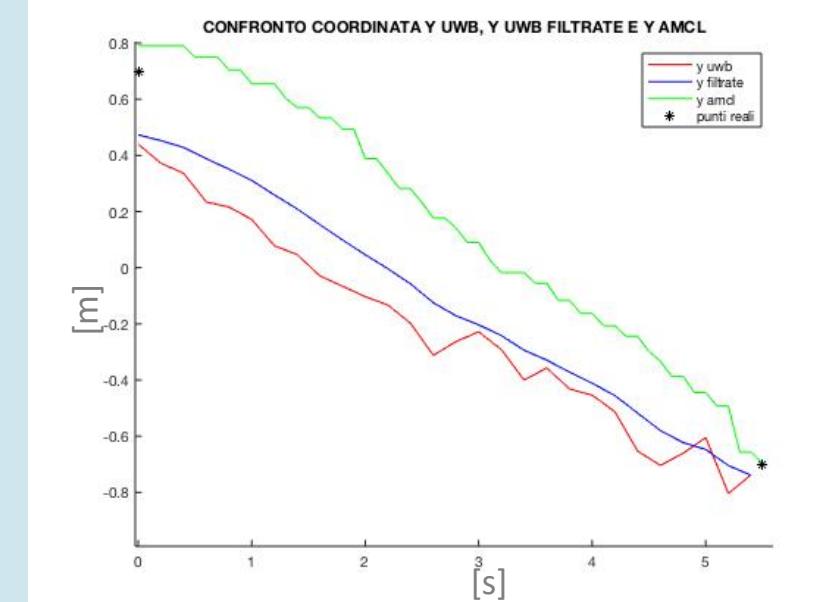
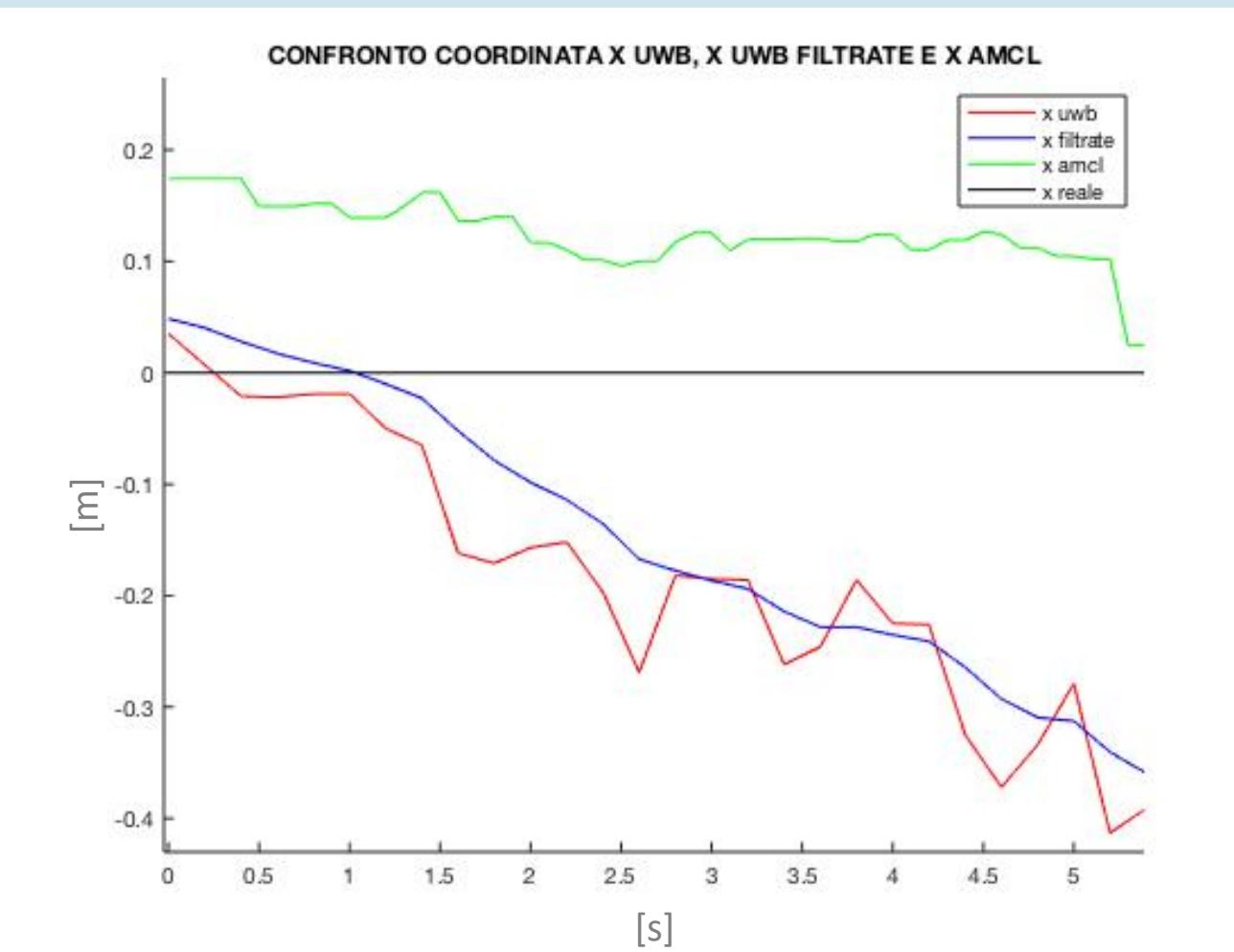




Spostamento lungo x costanti

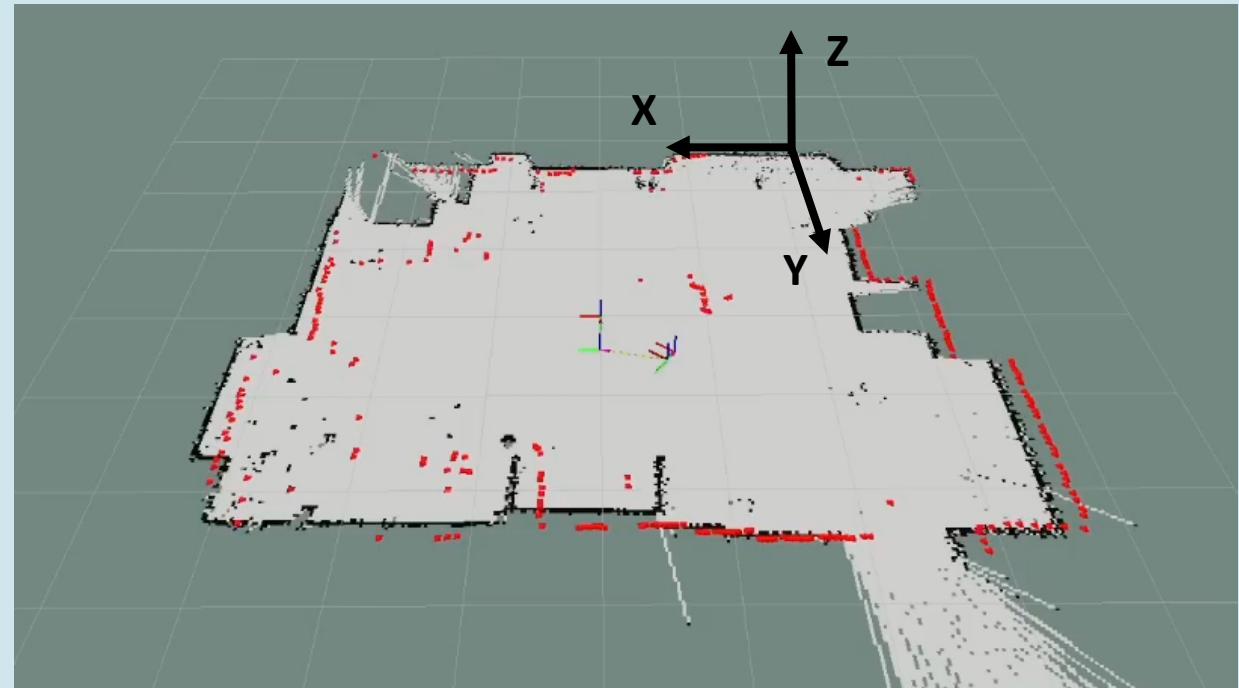
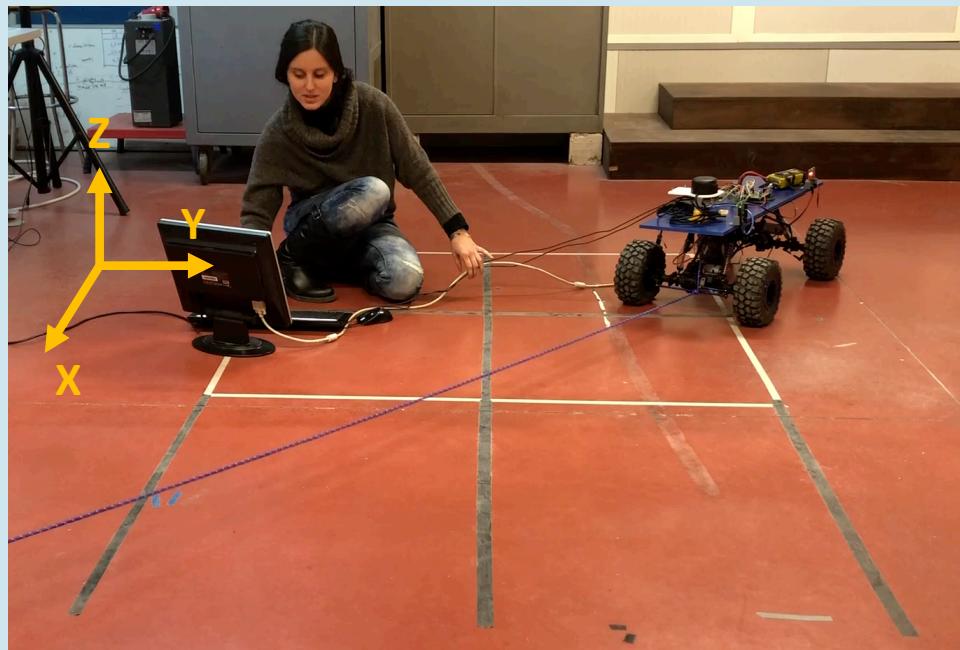
Il robot è stato fatto muovere con x costante pari a 0 m (nel frame *map*).

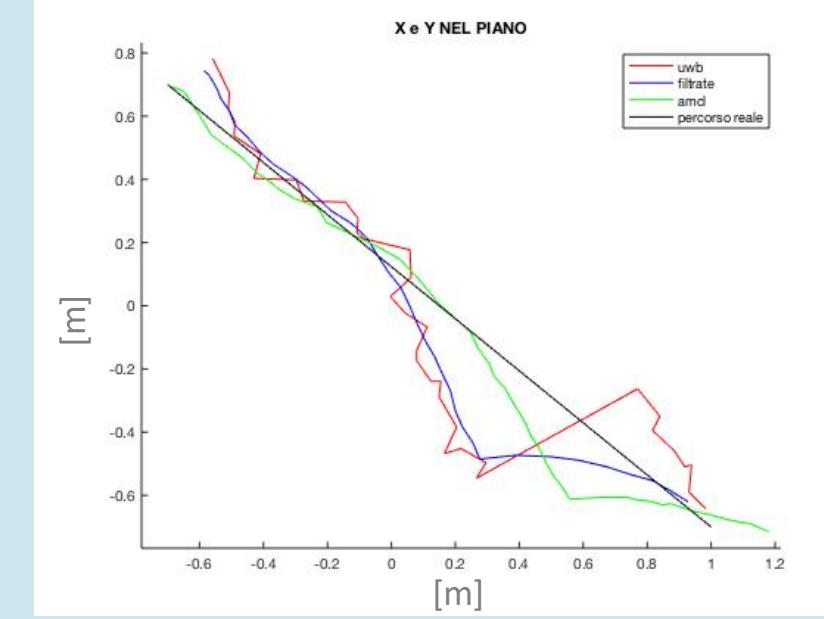
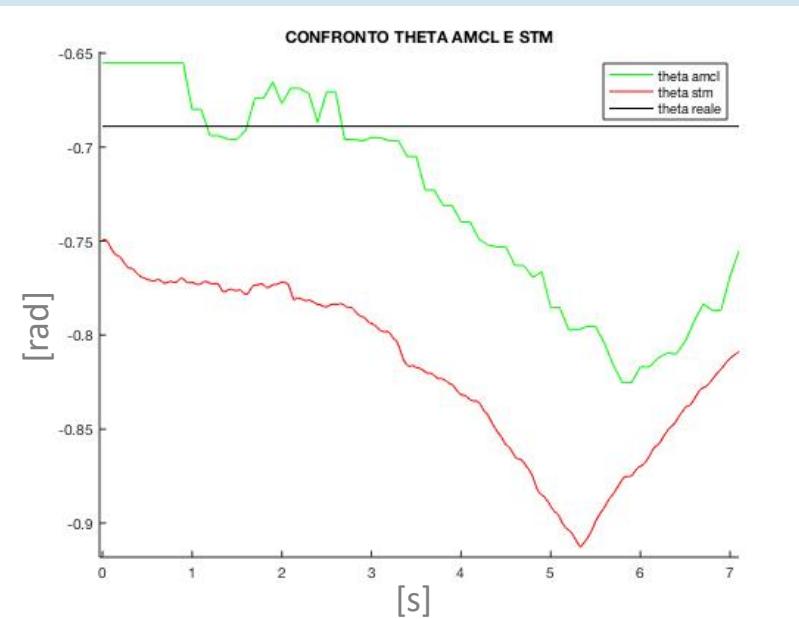
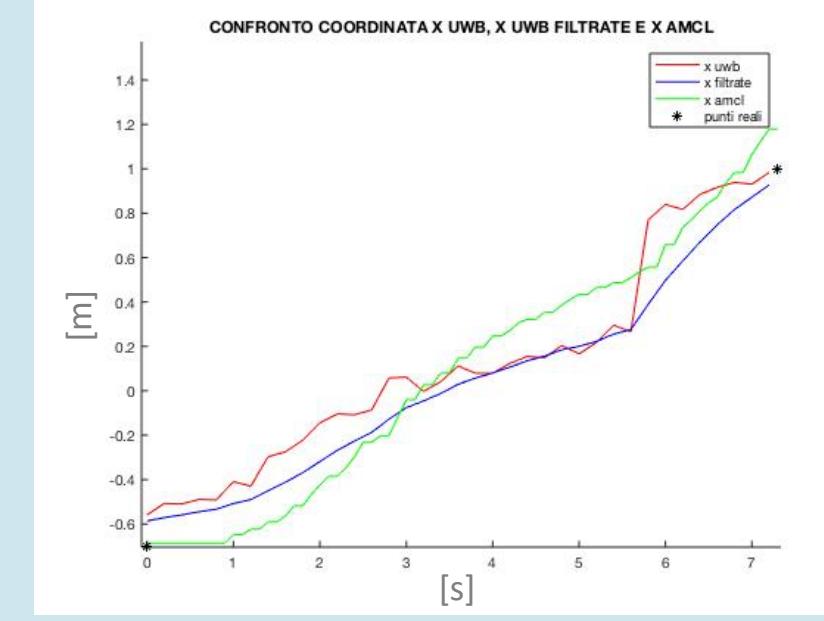
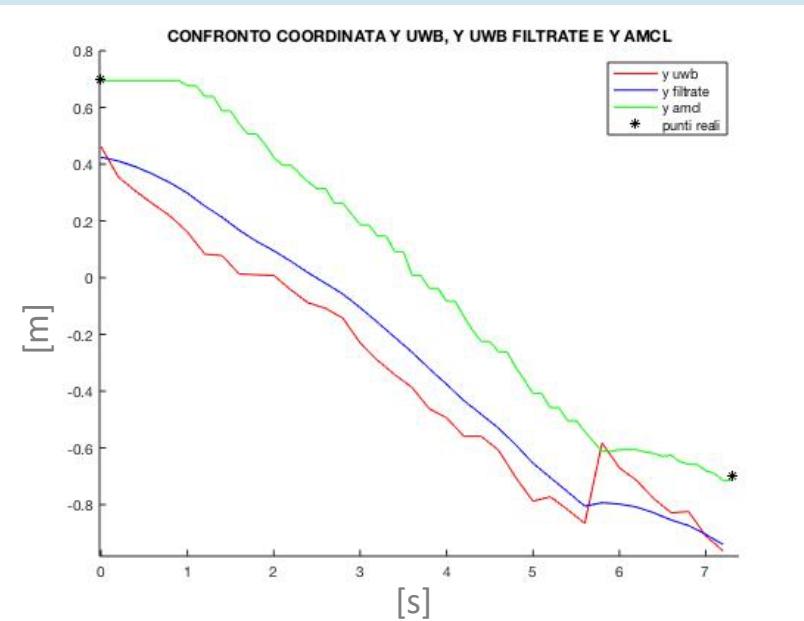




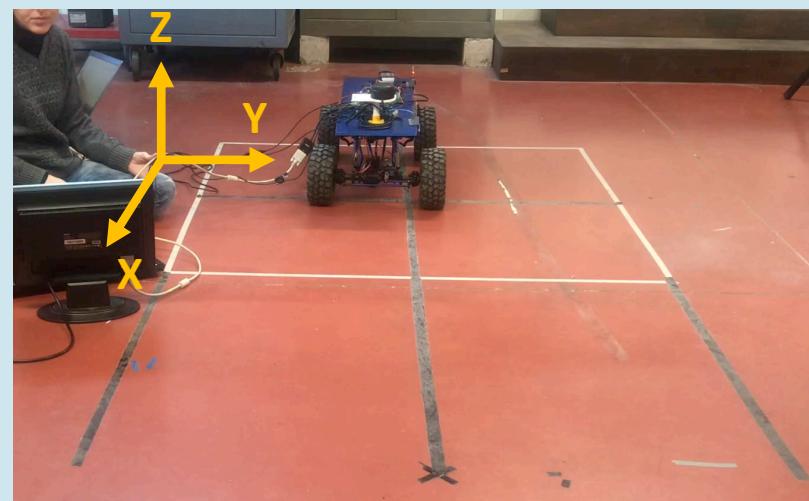
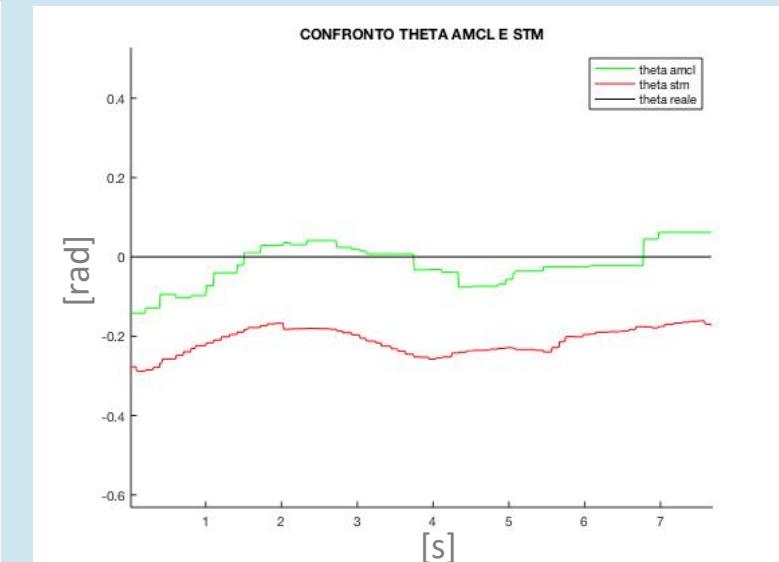
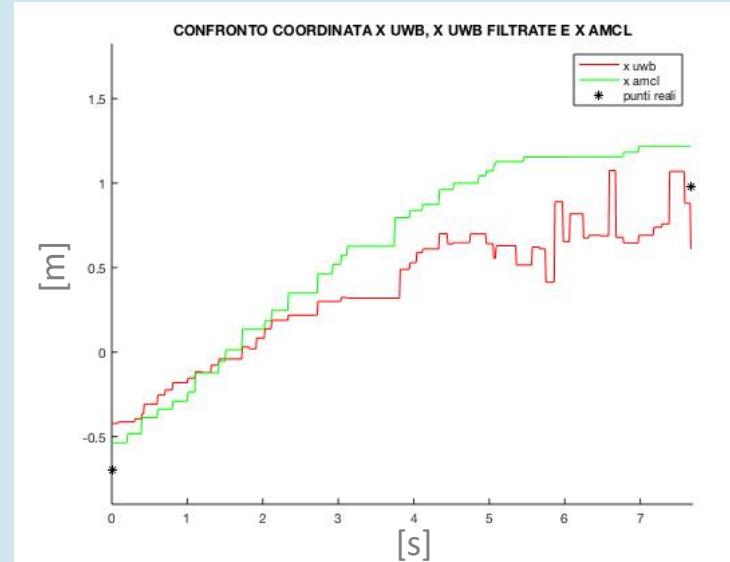
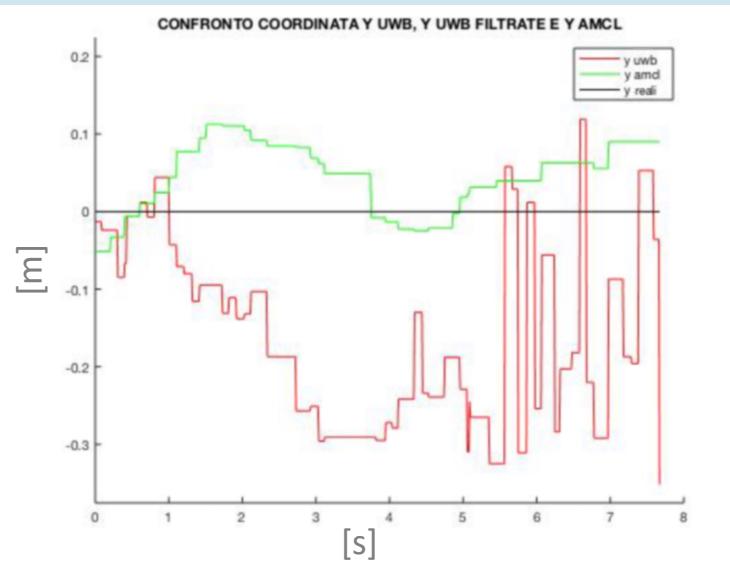
Spostamento lungo una diagonale

Il robot è stato fatto muovere dal punto $(-0.7\text{m}, 0.7\text{m})$ al punto $(1\text{m}, -0.7\text{m})$ nel frame *map*.

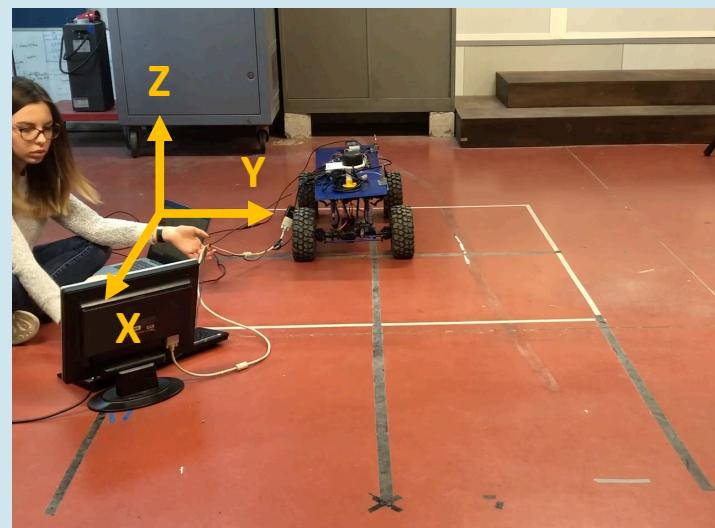
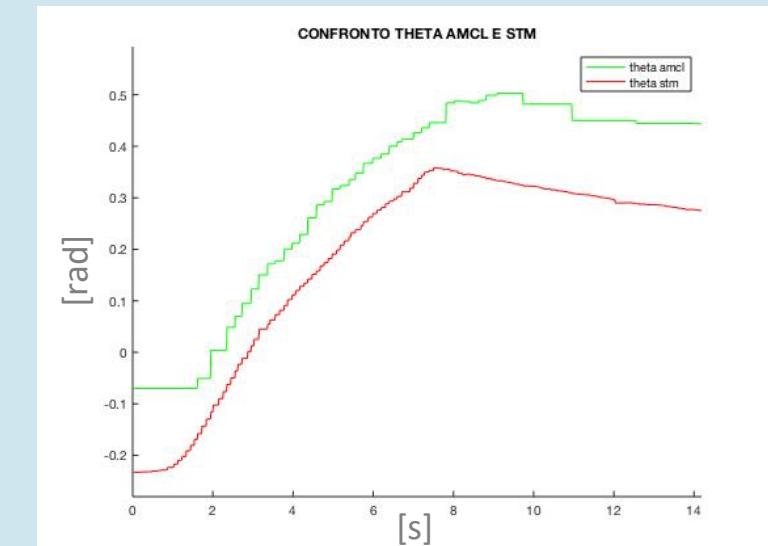
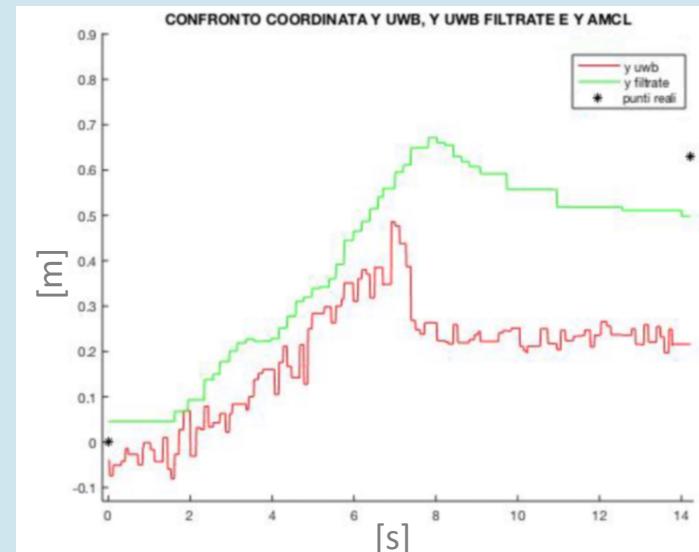
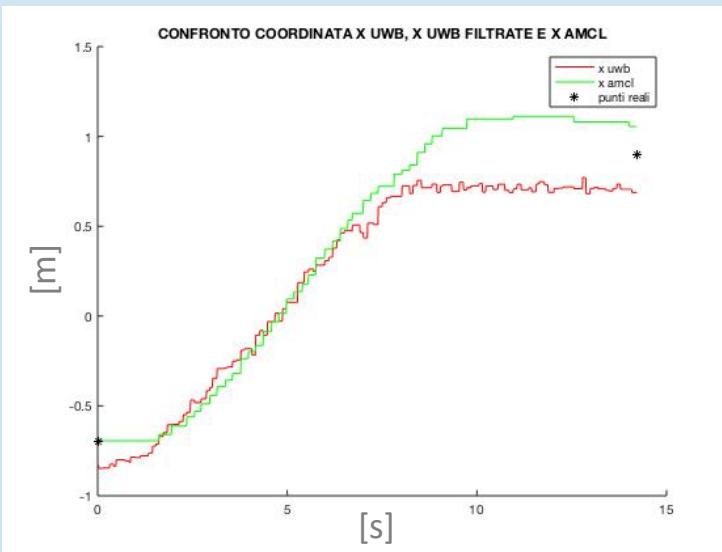




Raggiungimento waypoint frontale al robot



Raggiungimento waypoint a sinistra del robot





INTERFACCIA GRAFICA

File URDF

Unified Robot Description Format (URDF) è un formato XML che permette di descrivere la cinematica, la dinamica, la rappresentazione visiva e le caratteristiche di collisione del robot. Con questo formato è quindi possibile creare un modello 3D con caratteristiche di inerzia e limiti fisici. Per comporre le varie parti del modello vengono utilizzati dei tags :

- <robot>
- <link>
- <joint>

Il tag <robot> contiene al suo interno l'intera descrizione del modello, composto da un insieme di link e giunti.



```
<joint name="[name of the joint]" type="[type of the tag]">
  <parent link="parent_link"/>
  <child link="child_link"/>

  <axis ... />
  <calibration ... />
  <dynamics damping ... />
  <limit effort ... />
</joint>
```

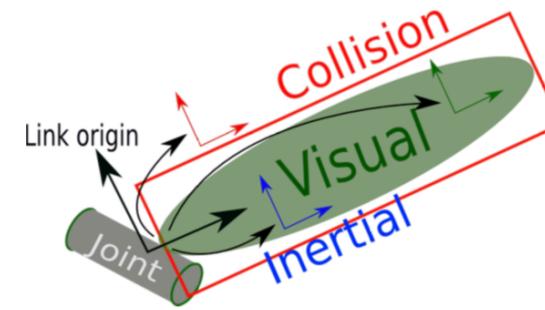
- ◀ <parent> represents parent link
- ◀ <child> represents parent link
- ◀ <axis> define the axis of rotation for revolute joints, the axis of translation for prismatic joints, and the surface normal for planar joints
- ◀ <limit> define lower and upper joint limit

Il tag `<joint>` rappresenta un giunto del robot, ed ha la funzione di connettere un link genitore ad un link figlio. È possibile specificare la cinematica e la dinamica del giunto ed anche i suoi limiti nel movimento. Ne esistono di varie tipologie come: fissi, continui e prismatici.



```
<link name="[name of the link]>
  <visual> ... </visual>
  <collision> ... </collision>
  <inertial> ... </inertial>
</link>
```

- ▶ <visual> represents the real link of the robot, and the area surrounding the real link is the <collision> section.
- ▶ <collision> encapsulates the real link to detect collision before hitting the real link
- ▶ <inertial> define the inertia of the link



Il tag <link> rappresenta un singolo link del robot. È possibile definire le sue caratteristiche: dimensione, forma, colore, proprietà dinamiche e di collisione.



File XACRO

I file URDF presentano dei limiti, infatti risultano essere molto complessi e ridondanti poiché non permettono di includere altri file URDF al loro interno, di utilizzare variabili o costanti o di utilizzare uno stesso blocco per descrivere più link. Ad esempio nel caso della definizione delle ruote di un veicolo non è possibile descrivere una sola volta le caratteristiche del link, per poi riutilizzarlo, ma ciascuna ruota deve essere caratterizzata completamente pur presentando le stesse caratteristiche delle altre.

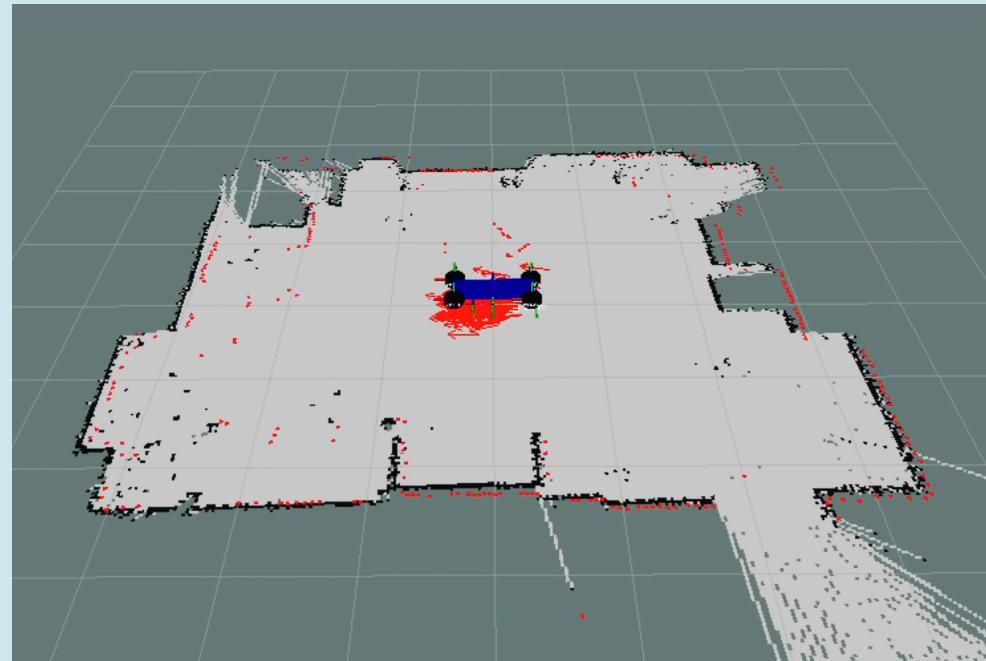
Tutti questi limiti, possono essere superati utilizzando l'estensione .xacro al posto di .urdf



Simulazione in Rviz

Il file .xacro permette di visualizzare la macchina in Rviz, simulando il suo spostamento.

Per fare ciò è necessario imporre una trasformata statica fra il corpo della macchina definito nel file URDF e il base_link di amcl.





MOVEBASE

movebase.cpp

Per implementare l'obstacle avoidance è stato utilizzato il pacchetto ros movebase che contiene il nodo **movebase.cpp**. Questo per funzionare ha bisogno di un waypoint, informazioni provenienti dal laser, di una mappa e della posizione del robot in essa. Restituisce in uscita il comando di velocità lineare e angolare che permetterebbe al robot di arrivare nel punto desiderato.

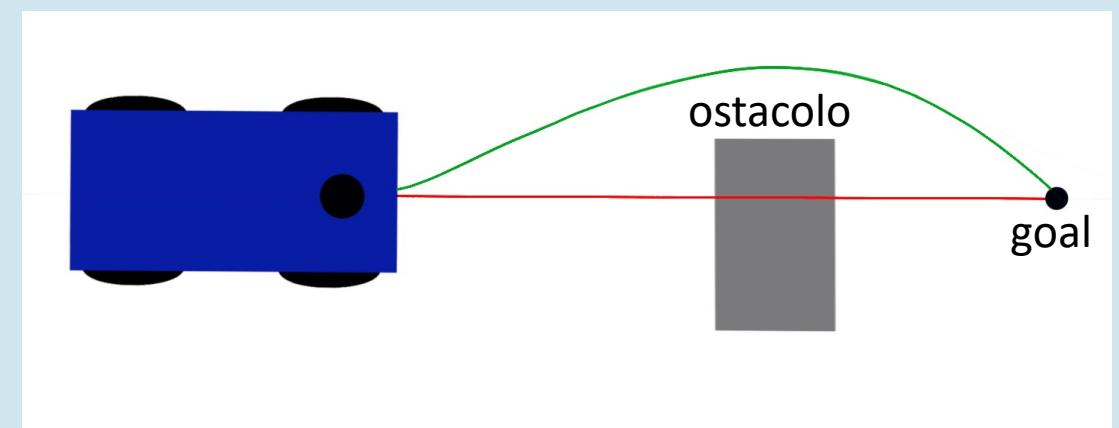
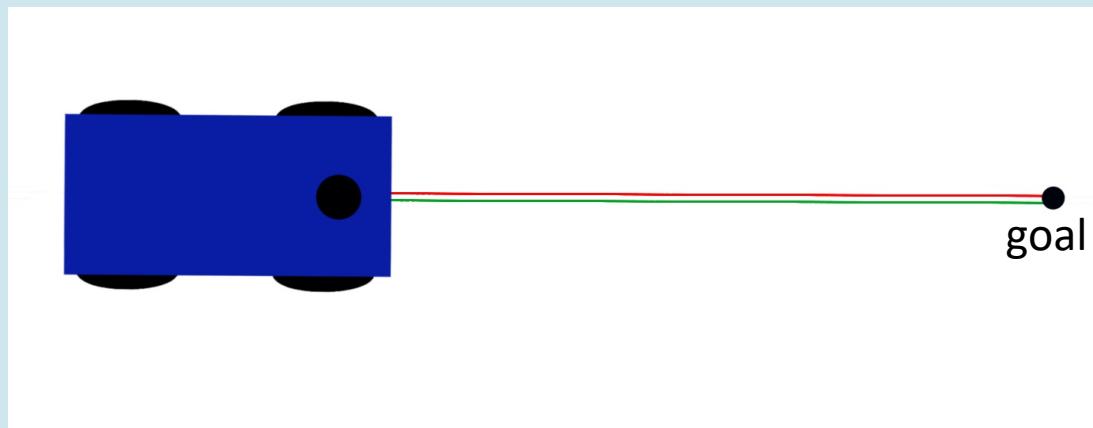
Per raggiungere il waypoint viene implementato nel nodo un algoritmo di pianificazione che sfrutta le informazioni provenienti da due costmap:

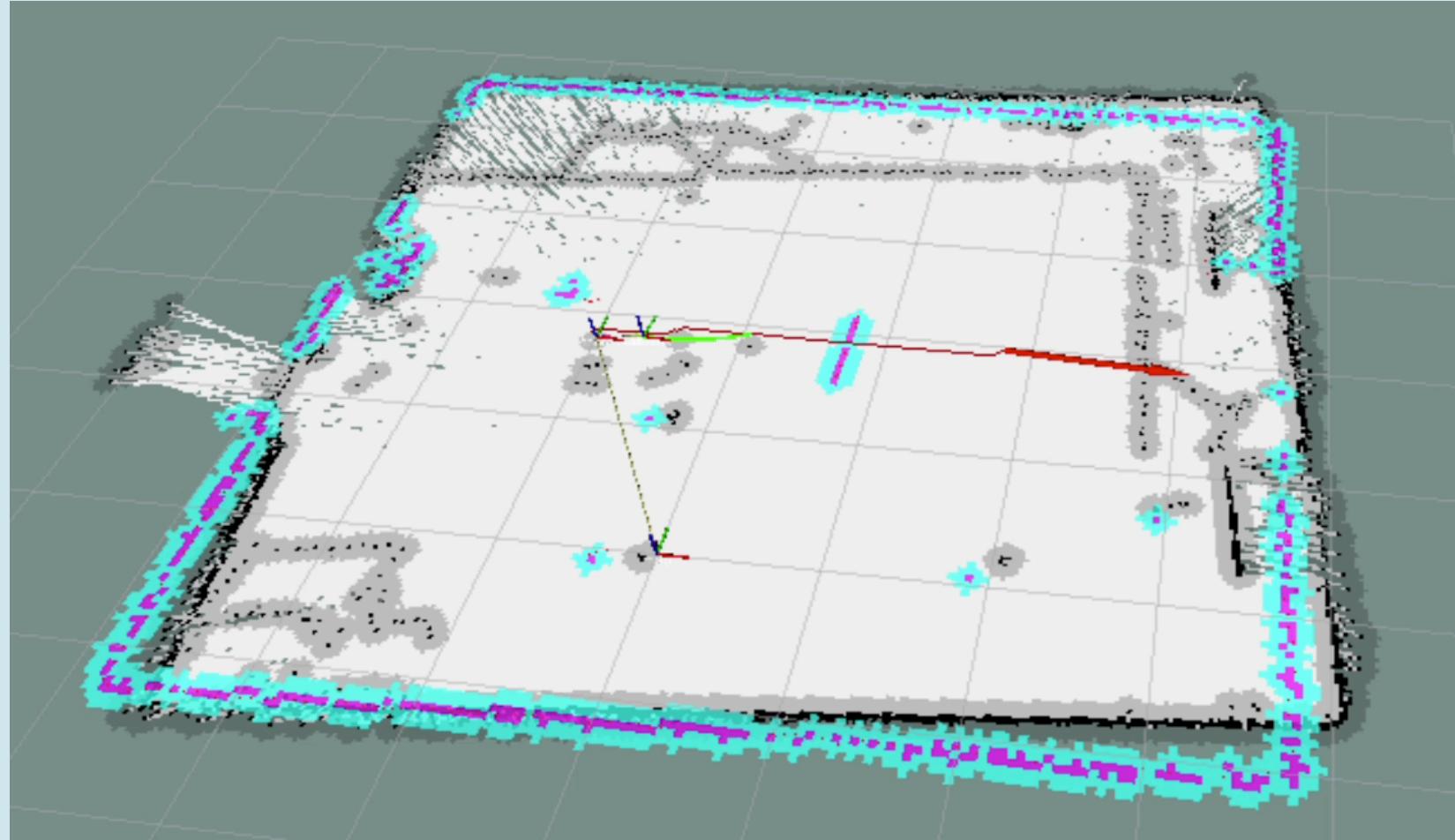
- **global costmap**: contiene le informazioni sugli ostacoli presenti nella mappa al momento della sua acquisizione
- **local costmap**: contiene le informazioni sugli ostacoli percepiti dal laser in tempo reale



Sono presenti due pianificatori di percorso:

- **global planner** : pianifica un percorso sulla global costmap che collega il robot al goal, utilizzando un algoritmo di ricerca su grafo (Dijkstra). (In rosso nelle immagini)
- **local planner**: pianifica un percorso sulla local costmap che cerca di seguire quello del global planner utilizzando un algoritmo basato sul campionamento. Il percorso viene aggiornato in tempo reale in base agli ostacoli percepiti dal laser. (In verde nelle immagini)





Esperimento in cui il robot è stato spostato manualmente per verificare l'effettivo aggiornamento dei percorsi e che il comando di velocità dato da movebase si annullasse una volta raggiunto il goal.

Nel video si può notare che quando il local planner (in verde) diverge dal global planner (in rosso), questo si aggiorna.



ACCORGIMENTI PER LAVORI FUTURI

- Integrazione di una IMU in fase di acquisizione della mappa, così da permettere al laser durante la scansione di poter ruotare intorno all'asse z e inclinarsi rispetto al piano.
- Utilizzo degli angoli di Roll e Pitch in uscita dal filtro AHRS, al fine di compensare durante la localizzazione eventuali inclinazioni del laser posizionato sul robot.
- Sviluppo del nodo movebase.cpp e implementazione di un controllo in velocità per il raggiungimento del goal.