



UNIVERSITÀ DI PISA

Dipartimento di Ingegneria dell'Informazione  
Corso di Laurea Ingegneria Robotica e dell'Automazione

Progetto di Sistemi di Guida e Navigazione

## Localizzazione mista di un veicolo attraverso sensori Lidar e Ultra-wideband

**Docente:**

Prof. Lorenzo Pollini

**Studenti:**

Do Won Park  
Luca Tedeschi  
Federico Viviani

Anno Accademico 2020/2021



# Indice

<b>Elenco delle figure</b>	<b>iii</b>
<b>Elenco delle tabelle</b>	<b>v</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Descrizione del sistema . . . . .	1
1.1.1 In dettaglio . . . . .	3
1.2 Punto di partenza . . . . .	6
1.2.1 Hardware . . . . .	6
1.2.2 Software . . . . .	7
1.3 Modifiche . . . . .	7
<b>2 Lavoro svolto</b>	<b>11</b>
2.1 Localizzazione . . . . .	11
2.1.1 Lidar . . . . .	11
2.1.2 UWB . . . . .	13
2.2 Comunicazione seriale . . . . .	15
2.2.1 Problematiche . . . . .	16
2.3 Localizzazione robusta . . . . .	22
2.3.1 Oscuramento Lidar . . . . .	24
<b>3 Dati e esperimenti</b>	<b>27</b>
3.1 Esperimento 1 - Fix di posa simulato con Rviz . . . . .	27
3.2 Esperimento 2 - Fix di posa durante navigazione in più stanze . . . . .	28
3.3 Esperimento 3 - Rapimento del robot . . . . .	30
3.4 Esperimento 4 - Navigazione outdoor . . . . .	33
<b>4 Conclusioni</b>	<b>35</b>

<b>5 Suggerimenti per sviluppi futuri</b>	<b>37</b>
<b>Bibliografia</b>	<b>39</b>
<b>Appendice A Cenni teorici</b>	<b>41</b>
A.1 Adaptive Monte Carlo Localization . . . . .	41
<b>Appendice B Guida al Codice</b>	<b>45</b>
<b>Appendice C Guida all'esperimento</b>	<b>57</b>

# Elenco delle figure

1.1	Schema hardware del veicolo . . . . .	2
1.2	Alimentazione . . . . .	3
1.3	Connessioni tra STM®e motori . . . . .	4
1.4	Schema completo . . . . .	5
1.5	Veicolo ricevuto dal gruppo precedente . . . . .	6
1.6	Confronto prima e dopo cablaggio . . . . .	7
1.7	PCB realizzata . . . . .	8
1.8	Veicolo finale . . . . .	9
2.1	Schema comunicazione seriale . . . . .	15
2.2	Misura tempo alto con oscilloscopio . . . . .	19
2.3	Modifica schematico Simulink per connessione Arduino e simulazione su PC . . . . .	20
2.4	Simulink - controllo iniziale . . . . .	21
2.5	Simulink - controllo finale . . . . .	22
2.6	Fix di posa . . . . .	23
2.7	Schema <code>amcl2robotpose</code> . . . . .	25
3.1	Esperimento 1 . . . . .	28
3.2	Esperimento 2 . . . . .	29
3.3	Esperimento 3 . . . . .	32
3.4	Posizionamento delle ancore UWB . . . . .	33
B.1	Grafico dell'analisi del tempo di regime dello yaw letto da STM . . . . .	49
B.2	Feedback all'utente dal sistema di salvataggio della posa relativa tra mappa e UWB . . . . .	51
B.3	Rimozione outlier . . . . .	52
C.1	Assi UWB . . . . .	58

---

C.2 Assi Lidar . . . . .	59
--------------------------	----

# Elenco delle tabelle

2.1	Parametri AMCL modificati . . . . .	12
2.2	Comunicazione seriale Intel®Joule™ → STM32® . . . . .	16
2.3	Comunicazione STM32® → seriale Intel®Joule™ . . . . .	16



# Capitolo 1

## Introduzione

L'obiettivo di questo progetto è quello di realizzare un veicolo alimentato a batteria in grado di localizzarsi su una mappa preacquisita e di navigare all'interno di essa sfruttando un sensore Lidar ed antenne Ultra-wideband (UWB).

Il motivo della scelta di tali sensori è legato al fatto che il veicolo deve esser capace di localizzarsi in una duplice maniera così da ovviare a fallimenti temporanei di uno dei due sistemi: nello specifico si utilizza il sensore Lidar per eseguire uno scan-matching rispetto alla mappa e si corregge la posa stimata, se necessario, sfruttando le informazioni provenienti dal sistema UWB. Infatti, i due metodi di localizzazione sono circa complementari, dato che lo scan-matching, una volta individuata la posizione approssimata nella mappa, è capace di dare stime corrette con elevata precisione, ma può arrivare a perdere completamente in ambienti troppo poco eterogenei (ad esempio un corridoio lungo e senza riferimenti) o in caso di oscuramento del sensore; è in questo caso che per correggere la stima di posa del robot subentrano le UWB.

### 1.1 Descrizione del sistema

L'hardware è composto da un veicolo a quattro ruote dotato di un motore per l'asse posteriore ed uno per l'asse anteriore, sul quale è presente anche un servo che funge da sterzo. Per quanto riguarda l'elettronica di bordo, sono presenti due unità centrali (vedi Figura 1.1):

- Un'Intel® Joule™, con sistema operativo Linux 16.04 su cui viene eseguito Robot Operating System (ROS)
- Un microcontrollore STM32F407® su cui è implementato il sistema di guida

Oltre a questi, sono presenti:

- Un sensore Lidar Slamtec® RPLIDAR-A3™
- Un sistema di localizzazione Ultra-wideband (Pozyx®) composto da due tag installate sul veicolo e 4 ancore disposte sul terreno.

A livello macroscopico, al modulo Intel® Joule™ è affidato il compito di raccogliere i dati provenienti dal Lidar e dalle tag, e fornire una stima della posa del robot all'interno di una mappa preacquisita; successivamente questa stima viene quindi inviata all'STM®, la quale calcola i valori di controllo in pulse-width modulation, PWM, per lo sterzo e l'acceleratore del veicolo affinché venga raggiunto un punto desiderato sulla mappa.

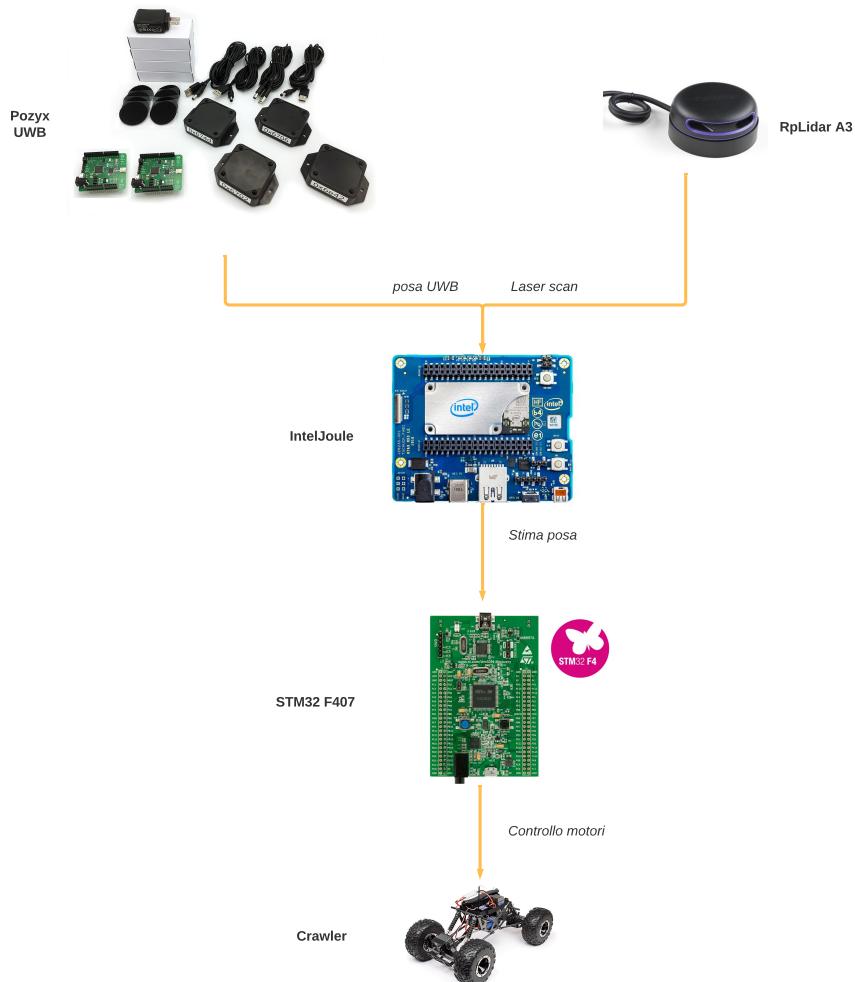


Figura 1.1 Schema hardware del veicolo

### 1.1.1 In dettaglio

#### Alimentazione

Il robot è dotato di due batterie:

- LiPo 14.6V 4200mAh, dedicata ad un'alimentazione generica, che viene sfruttata da tutti i componenti tranne i motori; da questa partono 3 linee di alimentazione:
  - a 14.6V per la STM<sup>®</sup>
  - a 12V per la Intel<sup>®</sup>Joule<sup>TM</sup>
  - a 5V per fornire alimentazione ausiliaria all'HUB-USB
- LiPo 7.4V 6000mAh dedicata ai motori

I vari convertitori di tensione sono tutti installati su una PCB, che è posizionata all'interno di un box metallico per schermare la radiazione elettromagnetica.

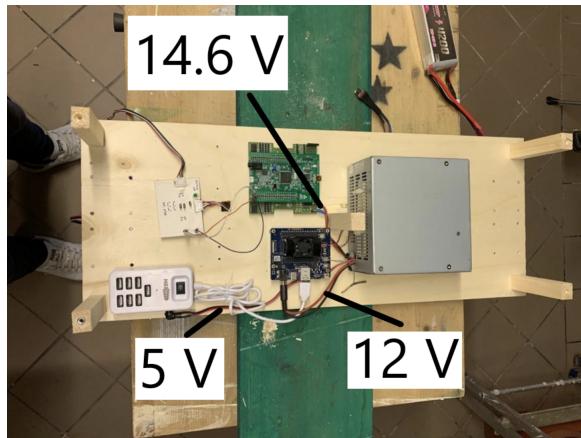


Figura 1.2 Linee di alimentazioni dalla LiPo 14.6V 4200mAh

#### Connessioni

Sia il Lidar che le due tag UWB sono connesse all'Intel<sup>®</sup>Joule<sup>TM</sup> tramite un HUB-USB, oltre a queste è presente un convertitore USB-TTL utilizzato per comunicare con l'STM32F407<sup>®</sup> via seriale. Il canale TX del convertitore TTL è connesso al canale RX della porta "UART 5" dell'STM<sup>®</sup> mentre il canale RX del convertitore TTL è connesso al canale TX della porta "UART 2" dell'STM.

### Connessioni in dettaglio tra STM®e Motori

I motori sono alimentati da una batteria LiPo dedicata da 7.4V 6000mAh, questa è connessa ad un controller che gestisce i due motori di traino anteriore e posteriore e fornisce alimentazione per il servo. Il controller dispone di una linea di ingresso, che si aspetta un segnale in PWM per regolare la velocità dei motori di traino, mentre il controllo dell'angolo del servo si ha tramite un secondo PWM separato. Lo schema di dettaglio è mostrato in Figura 1.3.

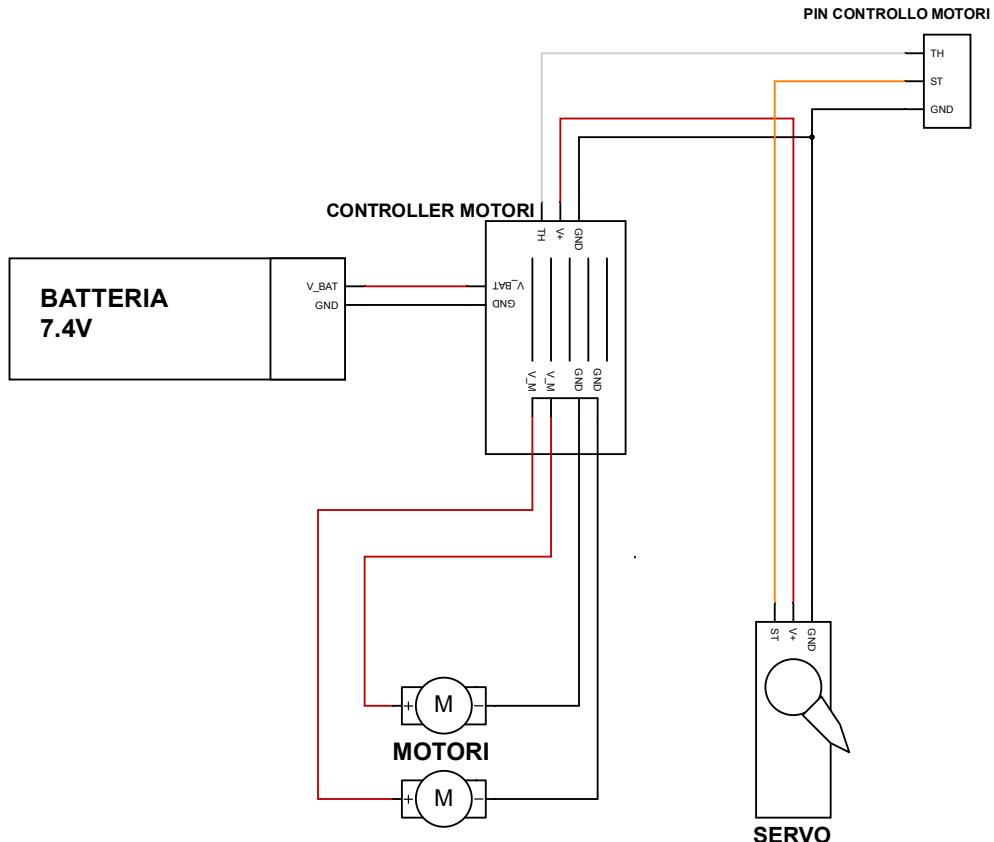


Figura 1.3 Connessione tra STM®e motori

### Connessioni in dettaglio - schema completo

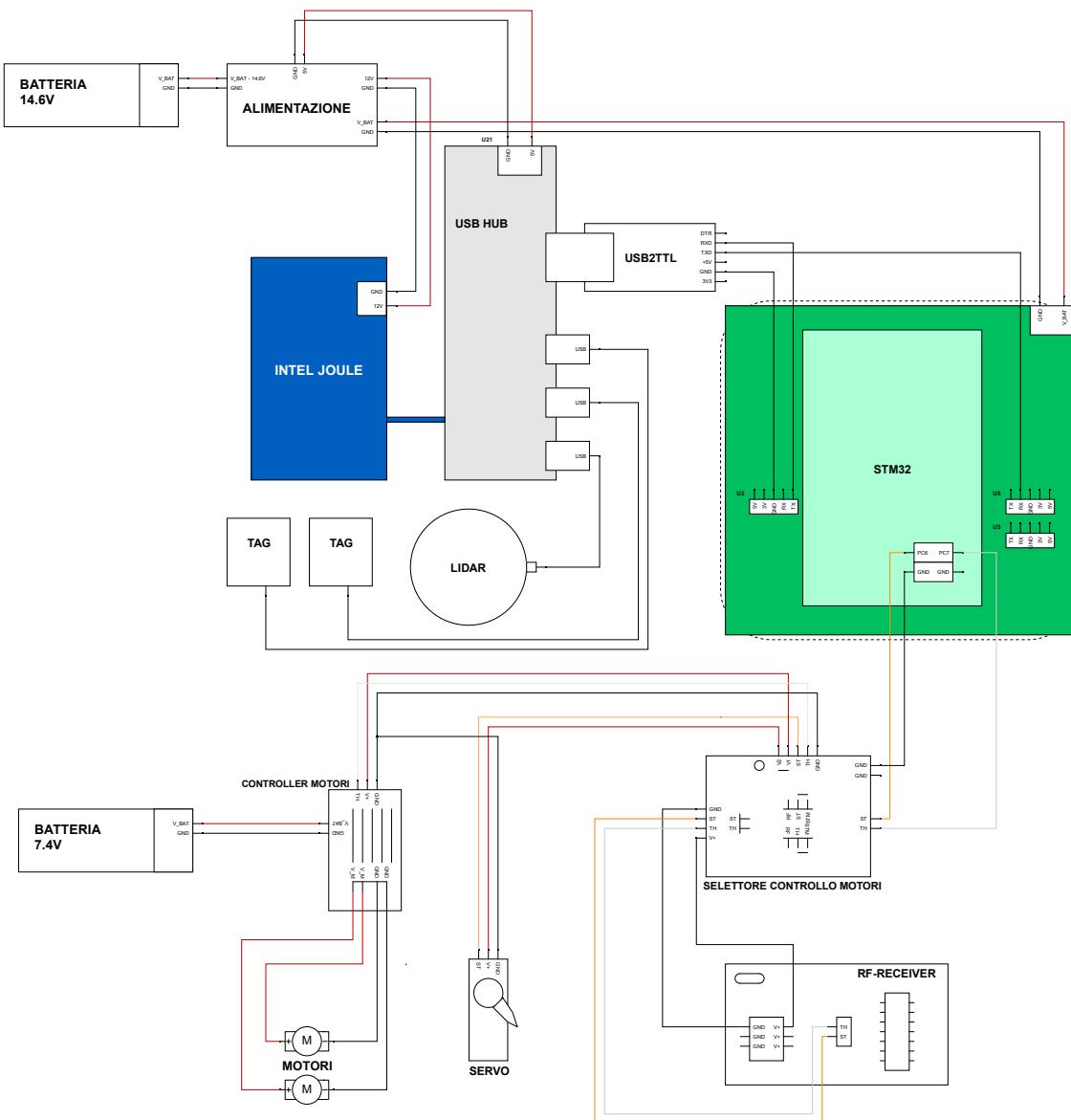


Figura 1.4 Schema completo

## 1.2 Punto di partenza

Il veicolo è stato ricevuto come mostrato in Figura 1.5.

Si è provato riprodurre i passaggi indicati dal gruppo precedente per valutare le condizioni correnti e identificare possibili comportamenti da correggere.

Dopo averlo ispezionato ed analizzato sono emerse alcune criticità sia a livello hardware sia a livello software e che verranno illustrate nei paragrafi seguenti.

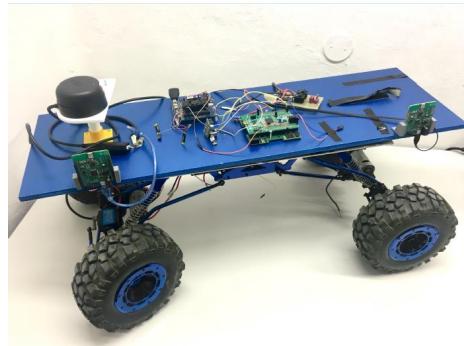


Figura 1.5 Veicolo di partenza, ricevuto dal gruppo precedente

### 1.2.1 Hardware

La prima problematica che è risultata evidente era riguardo la disposizione dell'elettronica sulla tavola in legno. Le schede non erano fissate ed i contatti tra le varie componenti risultavano molto precari. Ciò si traduceva in improvvisi malfunzionamenti o addirittura totali spegnimenti del sistema.

Per quanto riguarda la struttura, le sospensioni non erano in grado di mantenere la tavola con l'elettronica su di un piano orizzontale, ma si aveva un forte cedimento in particolar modo sul lato sinistro del veicolo, che causava un'inclinazione della tavola con conseguente impossibilità per il Lidar di effettuare un match con la mappa, dato che i piani non erano più allineati.

Inoltre, era necessario ripensare la disposizione generale delle varie componenti in modo da mantenere i sensori separati da tutto il resto, avere un maggiore livello di ordine ed anche una disposizione dei pesi più omogenea.

In secondo luogo vi era un problema di sottoalimentazione dei sensori, in particolare del Lidar, che non riceveva sufficiente alimentazione dalla sola porta USB-A della Intel® Joule™, ciò andava a compromettere i risultati della scansione, che in alcuni casi non era nemmeno in grado di avviarsi.

E' stato poi riscontrato un problema di interferenza elettromagnetica causato dalla batteria e dall'elettronica switching di alimentazione, che condizionavano in maniera significativa il funzionamento delle tag poste sul veicolo introducendo rumore non trascurabile e salti di posizione con ampiezza fino ad 1m.

### 1.2.2 Software

A livello software sono stati notati principalmente due problemi:

1. Servomotore, il quale non appena avviata la comunicazione seriale con la Intel® Joule™ iniziava ad avere un comportamento anomalo che consisteva in continui cambi di orientamento delle ruote
2. Il secondo problema invece si presentava nel publisher di posa delle UWB che era caratterizzato da un numero di fallimenti molto elevato, con un rapporto di circa un fallimento ogni 5 pacchetti inviati correttamente.

## 1.3 Modifiche

In primis è stato necessario ottenere un sistema robusto ed affidabile su cui effettuare i successivi esperimenti, con questo scopo sono stati installati dei distanziatori su cui sono state avvitate le schede, così da garantirne una posizione ben salda.

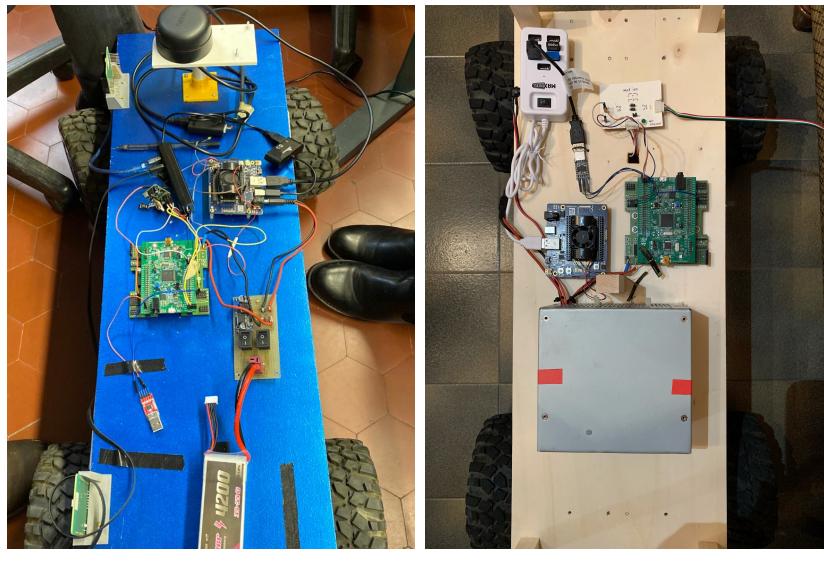


Figura 1.6 Confronto del cablaggio prima e dopo

Si è poi proceduto ad un rifacimento totale dei cablaggi, dato che era presente un disordine eccessivo, con cambi di colorazione, svariate giunzioni e connessioni parziali. Per raggiungere un livello di ordine maggiore e migliorare la facilità di assemblaggio sono state adottate delle piattine a 5 fili per le connessioni ed è stata sviluppata anche una piccola PCB che consente di utilizzare, alternativamente all'STM®, un radiocomando per il controllo di sterzo e acceleratore (vedi Figura 1.6).

La piccola PCB prodotta (Figura 1.7) ha un led di stato verde che notifica quando l'alimentazione dei motori è attiva; inoltre sono presenti due jumper che permettono di scegliere tra STM® e radiocomando come sorgente del controllo per sterzo e acceleratore. Sono sempre disponibili all'utente i contatti per potersi connettere e tramite telemetria leggere quanto prodotto dal radiocomando.

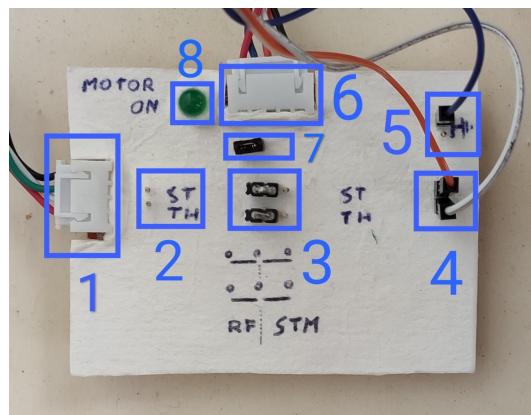


Figura 1.7 PCB realizzata

1. Connettore per il ricevitore del radiocomando;
  2. Pin per prelievo segnali PWM (steering/throttle) provenienti dal radiocomando;
  3. Jumper di selezione per la fonte di controllo: (jumper a sinistra) controllo da radiocomando e (jumper a destra) controllo da STM;
  4. Pin per connessione dei canali PWM provenienti da STM® (steering/throttle);
  5. Pin per connessione GND comune;
  6. Connettore per i motori;
  7. Jumper di abilitazione alimentazione motori;
  8. LED di stato, indica quando l'alimentazione dei motori è attiva.

Per risolvere il problema di sottoalimentazione dei sensori è stato acquistato un HUB-USB con possibilità di alimentazione esterna; è stato quindi aggiunto un convertitore DC-DC nella scheda di alimentazione per avere a disposizione una linea a 5V, a cui è stato connesso un cavo con apposito spinotto, per permettere l'alimentazione dell'HUB-USB dalla batteria.

Per quanto riguarda la struttura meccanica, la durezza delle sospensioni è stata aumentata al massimo possibile, riducendo sensibilmente l'inclinazione del veicolo in curva.

Anche la struttura di supporto dell'elettronica è stata completamente rivista giungendo ad un rifacimento totale: infatti, a seguito della scoperta dell'interferenza elettromagnetica sulle tag da parte della sezione di alimentazione, si è scelto di disporre i sensori su un livello separato rispetto a quello delle schede di controllo e di alimentazione. Pertanto, è stata adottata una nuova struttura in compensato, che si sviluppa su due piani paralleli disposti a 10cm di distanza; al livello più basso si trovano Intel® Joule™ ed STM®, con la scheda di alimentazione che è stata schermata inserendola all'interno di un box metallico, mentre al livello superiore vi è il Lidar in posizione centrale, e le due tag disposte lateralmente. Tutte le batterie adesso si trovano al di sotto del piano in compensato, così da concentrare il peso del veicolo in basso, ottenendo un baricentro più favorevole.

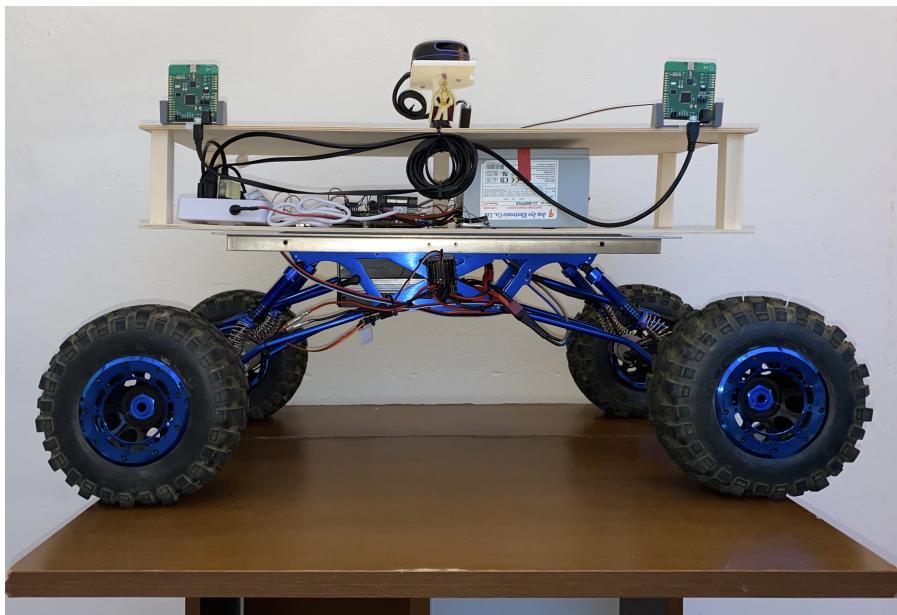


Figura 1.8 Veicolo Finale



# Capitolo 2

## Lavoro svolto

### 2.1 Localizzazione

L'obiettivo principale di questo lavoro consisteva nella sintesi di un sistema di localizzazione capace di sfruttare in modo complementare il sensore Lidar ed il sistema Pozyx® al fine di ottenere un robot capace di localizzarsi in modo affidabile all'interno di una mappa preacquisita. Il problema da risolvere è quello tipico di ogni sistema che sfrutta la tecnologia lidar affiancata ad un filtro a particelle per la localizzazione: infatti, possono verificarsi casi in cui l'insieme di stati probabili non si concentra a sufficienza, casi in cui si evidenziano più zone equiprobabili causando un'indeterminazione non facilmente risolvibile, e casi in cui l'assenza di punti di riferimento validi si traduce nella totale incapacità di percepire spostamenti. L'unico modo di risolvere tali problematiche è quello di "suggerire" all'algoritmo la posa approssimata, anche con varianza "grande", affinché il filtro a particelle concentri la nuvola di stati probabili circa nella giusta posizione.

#### 2.1.1 Lidar

Il Lidar si è rivelato essere il sensore più affidabile a disposizione, sia per quanto riguarda l'accuratezza dei risultati, sia per quanto riguarda la percentuale di fallimenti, che è prossima allo zero.

Basandosi su questo presupposto e sull'approccio utilizzato nel progetto precedente, l'utilizzo del Lidar è stato il punto di partenza del lavoro.

L'algoritmo di localizzazione designato è l'Adaptive Monte Carlo Localization (AMCL) descritto nell'Appendice A.1.

Nelle condizioni in cui il robot è stato ricevuto, l'algoritmo non produceva risultati soddisfacenti, il match con la mappa era scarso e di conseguenza anche la localizzazione del veicolo su di essa.

Effettuando un'analisi approfondita del flusso di dati del sistema, è stato appurato che la causa principale di una localizzazione poco accurata risiedeva nella fonte dell'odometria, necessaria ad AMCL sotto forma di `tf` per funzionare correttamente; questa era ottenuta dai dati di posa provenienti dal sistema UWB, le cui scarse performance si riflettevano in una odometria di altrettanto scarsa qualità. Ripartendo dunque da un AMCL in versione originale, effettuando tuning su alcuni dei suoi parametri e optando per il nodo `laser_scan_matcher` che utilizza il Lidar stesso come sorgente di odometria, il risultato è notevolmente migliorato rispetto a quello precedente. Infatti, osservando il robot tramite `Rviz`, partendo da una posa iniziale caratterizzata da una certa covarianza, è sufficiente uno spostamento di poche decine di centimetri per arrivare a convergenza.

I parametri modificati sono mostrati in Tabella 2.1.

Tabella 2.1 Parametri AMCL modificati

Parametro	Default	Modificato	Descrizione
<code>kld_err</code>	0.01	0.03	Massimo errore tra la vera distribuzione e quella stimata.
<code>laser_z_hit</code>	0.95	0.99	Peso della parte <code>z_hit</code> del modello.
<code>laser_z_rand</code>	0.05	0.01	Peso della parte <code>z_rand</code> del modello.
<code>update_min_d</code>	0.2	0.05	Movimento traslazionale richiesto prima di eseguire un aggiornamento del filtro.
<code>update_min_a</code>	$\pi/6$	0.01	Movimento rotatorio necessario prima di eseguire un aggiornamento del filtro.

Tuttavia, nonostante i buoni risultati in condizioni di funzionamento normali, cioè con robot che naviga all'interno di una stanza della quale è disponibile la mappa e nessun tipo di interferenza con il sensore Lidar, ciò che si desidera è rendere la localizzazione robusta ad eventuali perturbazioni o fallimenti. Infatti, nel caso in cui il sensore Lidar non riuscisse ad effettuare scansioni per un certo intervallo di tempo, oppure sbagliasse la localizzazione del robot a causa di errori nel matching della mappa, si rende necessario un meccanismo che si accorga di tale errore e che intervenga per correggerlo. Tale meccanismo, che verrà approfondito nel paragrafo 2.3, è stato messo

a punto sfruttando le informazioni provenienti dalle UWB per mandare dei fix di posa quando vi è un'eccessiva differenza rispetto alle stime di posizione di AMCL, scegliendo quella fornita dalle UWB come base su cui reinizializzare il filtro.

### 2.1.2 UWB

La prima cosa da fare per sfruttare le UWB, dopo averle posizionate in un nuovo setup, è attivare la procedura di autocalibrazione, in modo che le ancore si identifichino automaticamente e calcolino le distanze l'una dall'altra. Ciò può esser fatto tramite il bash script `./autocalibration.sh` che una volta terminato, oltre a settare in modo corretto le ancore, mostra a schermo l'esito della procedura, in modo da poter valutare la bontà dei risultati. Per eventuali approfondimenti fare riferimento a Crosato-Tesconi[1].

Conclusa la fase di setup e calibrazione inziale, è possibile sfruttare i dispositivi Pozyx® connessi al sistema tramite dei nodi dedicati, essenzialmente publisher di posa. Per lo sviluppo di questa parte del software si è deciso di partire dagli scripts Python messi a disposizione dalla Pozyx® ([2]) e di adattarli poi in base alle esigenze del progetto: sono stati realizzati vari nodi che pubblicano la posa delle tag (posizione+quaternione) e le relative `tf`. Dato il rumore presente, è stata implementata la possibilità di filtrare i dati ottenuti dalle tag con filtri a media mobile, tuttavia si sconsiglia di utilizzarli, vista l'introduzione di un ritardo. Inoltre, è da sottolineare che l'utilizzo della media semplice è molto sensibile alla presenza di outliers che sono stati osservati frequentemente durante la conduzione degli esperimenti.

Per approfondimenti sui nodi fare riferimento ad Appendice B.

Fin dai primi test è sempre emersa una certa inaffidabilità delle tag come principale sistema di localizzazione, poiché pubblicano dati fin troppo rumorosi ed sono soggette a improvvisi fallimenti.

Ad oggi sono state identificate e risolte due possibili cause: per quanto riguarda la prima, si trattava di un tag difettoso che è stato sostituito, mentre la seconda era una forte interferenza elettromagnetica proveniente dall'alimentazione e relativa parte switching, che è stata arginata racchiudendo la parte switching all'interno di un box metallico e spostando la batteria al livello inferiore del robot.

E' stato predisposto anche un foglio di alluminio che può essere posizionato come ulteriore strato isolante, ma non è sembrato portare alcun tipo di beneficio.

Nonostante le soluzioni adottate, i dati che si ricevono continuano ad essere non precisi e di conseguenza non utilizzabili in modo continuo per una localizzazione affidabile.

Per questo motivo si è deciso di sfruttare il sistema UWB parallelamente ai risultati dello scan matching, utilizzando il dato di posa solo per effettuare un fix di posizione quando vi è una differenza superiore ad 1.5m rispetto a quella fornita da AMCL (vedi paragrafo [2.3](#)).

## 2.2 Comunicazione seriale

Il sistema che si occupa della comunicazione seriale è stato parzialmente riscritto per ottenere maggiore chiarezza e ordine. Lo scopo è quello di gestire il canale di comunicazione tra Intel® Joule™ ed STM® per lo scambio reciproco di dati attraverso le porte seriali. L'Intel® Joule™, funge da raccordo con l'ambiente ROS leggendo e scrivendo su topic accessibili dal resto del sistema, mentre l'STM® crea una connessione con lo schematico Simulink® su di essa implementato.

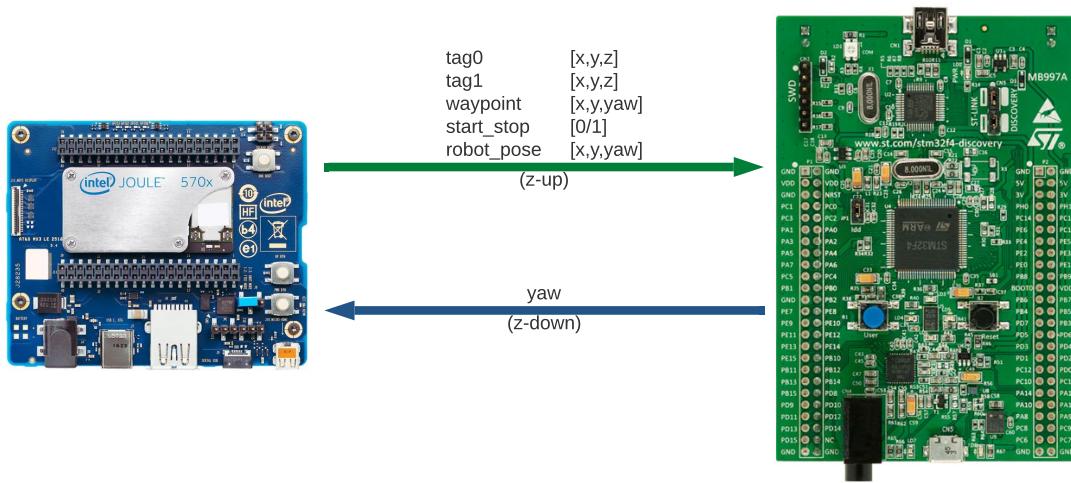


Figura 2.1 Schema comunicazione seriale

I dati necessari per la localizzazione vengono raccolti a livello dell'Intel® Joule™ , che procede quindi ad elaborarli ed in seguito inviarli alla scheda a valle (STMF407®) insieme al goal da raggiungere<sup>1</sup>. L'STM®, sulla base dei dati ricevuti, calcola il controllo da dare ai motori per raggiungere l'obiettivo impostato; parallelamente a ciò misura costantemente l'heading del veicolo inviandolo all'Intel® Joule™ .

I dati scambiati sono riassunti nelle Tabelle 2.22.3.

I primi tre elementi sono header che codificano il tipo di messaggio, mentre i restanti sono informazioni utili che verranno lette ed interpretate.

<sup>1</sup>impostabile tramite Rviz con il comando `2DNavGoal` oppure pubblicando sul topic `/waypoint_publisher`

<sup>2</sup>float

Tabella 2.2 Comunicazione seriale Intel® Joule™ → STM32®

Nome	Descrizione	Dimensione
HEADER_A	0x1A	1 Byte
HEADER_B	0x1B	1 Byte
PAYLOAD	0x2C	1 Byte
pos_x_f	tag 0 - coordinata $x$ - frame UWB	4 Byte <sup>2</sup>
pos_y_f	tag 0 - coordinata $y$ - frame UWB	4 Byte <sup>2</sup>
pos_z_f	tag 0 - coordinata $z$ - frame UWB	4 Byte <sup>2</sup>
pos_x_b	tag 1 - coordinata $x$	4 Byte <sup>2</sup>
pos_y_b	tag 1 - coordinata $y$	4 Byte <sup>2</sup>
pos_z_b	tag 1 - coordinata $z$	4 Byte <sup>2</sup>
way_x	waypoint - coordinata $x$ - frame map	4 Byte <sup>2</sup>
way_y	waypoint - coordinata $y$ - frame map	4 Byte <sup>2</sup>
way_z	waypoint - yaw - frame map	4 Byte <sup>2</sup>
start_stop	0 = disabilita motori   1 = abilita motori	4 Byte <sup>2</sup>
robot_pose_x	posa stimata da AMCL - coordinata $x$ - frame map	4 Byte <sup>2</sup>
robot_pose_y	posa stimata da AMCL - coordinata $y$ - frame map	4 Byte <sup>2</sup>
robot_pose_z	posa stimata da AMCL - yaw - frame map	4 Byte <sup>2</sup>

In dettaglio, le coordinate delle due tag vengono utilizzate dall'STM® per misurare lo yaw del veicolo<sup>3</sup>, che insieme alla posa stimata da AMCL, ed al waypoint impostato, sono sfruttate per determinare il controllo dei motori. Infine l'elemento **start\_stop** (booleano) li abilita o disabilita.

Il valore dello *yaw* è inviato dall'STM® in un pacchetto come quello mostrato in Tabella 2.3.

Tabella 2.3 Comunicazione STM32® → seriale Intel® Joule™

Nome	Descrizione	Dimensione
HEADER_A	0x1A	1 Byte
HEADER_B	0x1B	1 Byte
PAYLOAD_POSE	0x2C	1 Byte
yaw	$yaw$	4 Byte

## 2.2.1 Problematiche

A seguito della prima prova sperimentale in cui si chiedeva al robot di portarsi da una punto di partenza ad uno di arrivo, si è osservato un malfunzionamento del sistema:

---

<sup>3</sup>L'angolo è misurato rispetto all'asse  $x$  del frame UWB, riportandolo poi in un sistema di riferimento  $z-down$

non appena i motori venivano abilitati, il servo iniziava ad avere un comportamento anomalo variando continuamente tra l'orientazione corretta e la posizione di riposo. Questo tipo di comportamento comprometteva la possibilità di portare a termine test e verifiche sul software sviluppato. Sicuramente il difetto era presente anche nel sistema originale consegnato insieme al robot, tuttavia, le problematiche presenti a monte del controllo lo avevano mascherato impedendone l'identificazione prima della loro soluzione, poiché sembravano esserne la causa.

Si è resa quindi più che necessaria l'identificazione della sorgente del problema.

La ricerca è avvenuta in più passi, con lo scopo di restringere volta per volta il malfunzionamento ad un'area sempre più circoscritta:

- **Test 1** - isolamento del controllo: in questa prova è stato testato il corretto funzionamento di tutti i motori (anteriore, posteriore e servo) connettendoli direttamente ad un segnale PWM noto generato con un Arduino®, con andamento ad onda triangolare. Il test ha avuto esito positivo restringendo la zona in cui cercare il difetto alla coppia Intel® Joule™ ed STM® ed alla loro connessione e comunicazione;
- **Test 2** - controllo dei dati provenienti dalla Intel® Joule™ : in questa prova è stato chiuso su se stesso il sistema di comunicazione seriale dell'Intel® Joule™ connettendo il canale di trasmissione a quello di ricezione ed osservando i dati ricevuti. Anche questo test ha avuto esito positivo escludendo la scheda dalle possibili sorgenti del malfunzionamento;
- **Test 3** - controllo ricezione seriale STM®: in questa prova si è voluto verificare che il problema non fosse la comunicazione seriale tra Intel® Joule™ e STM®, generando gli stessi messaggi che sarebbero stati prodotti dall'Intel® Joule™ con un Arduino ed inviandoli quindi all'STM®. Il servo ha mantenuto il comportamento indesiderato oscillando tra le solite posizioni, confermando di conseguenza che la sorgente del problema risiedeva all'interno dell'STM®;
- **Test 4** - controllo dei segnali in uscita dall'STM®: questo test è stato svolto mantenendo Arduino® come sorgente del messaggio seriale contenente dati costanti, a cui dovrebbe corrispondere un controllo costante. I due segnali PWM di controllo dei motori sono stati dunque intercettati ed analizzati tramite un

oscilloscopio digitale, grazie al quale è stato possibile osservare una variazione sul tempo di livello alto del segnale in ingresso al servo. I due valori di “tempo alto” osservati e mostrati in Figura 2.2, codificano esattamente la posizione di estrema rotazione verso destra (2.21ms) e quella di riposo centrale (1.54ms) descritte in precedenza. Figura 2.2;

- **Test 5** - controllo dei segnali in uscita dall’STM® a livello dello schematico Simulink®: stabilito che la causa del malfunzionamento era da imputarsi al software implementato nell’STM®, è stato modificato il punto di ingresso dei dati nello schematico predisponendolo a leggere dalla porta USB di un computer alla quale connettere Arduino®. Lo schematico è mostrato in Figura 2.3: è stato inserito un subsystem per interfacciare lo streaming dei dati sulla porta seriale, costituito da pacchetti di 52 byte, con la restante parte del sistema che si aspetta 13 elementi da 4 byte ciascuno. Eseguendo quindi la simulazione su computer è stato possibile andare ad analizzare, uno ad uno, i valori dei vari segnali presenti, cercando il punto in cui avveniva la corruzione dei dati. L’analisi condotta non ha portato a risultati tangibili mostrando un’apparente corretto funzionamento del sistema.



Figura 2.2 Misura tempo alto con oscilloscopio

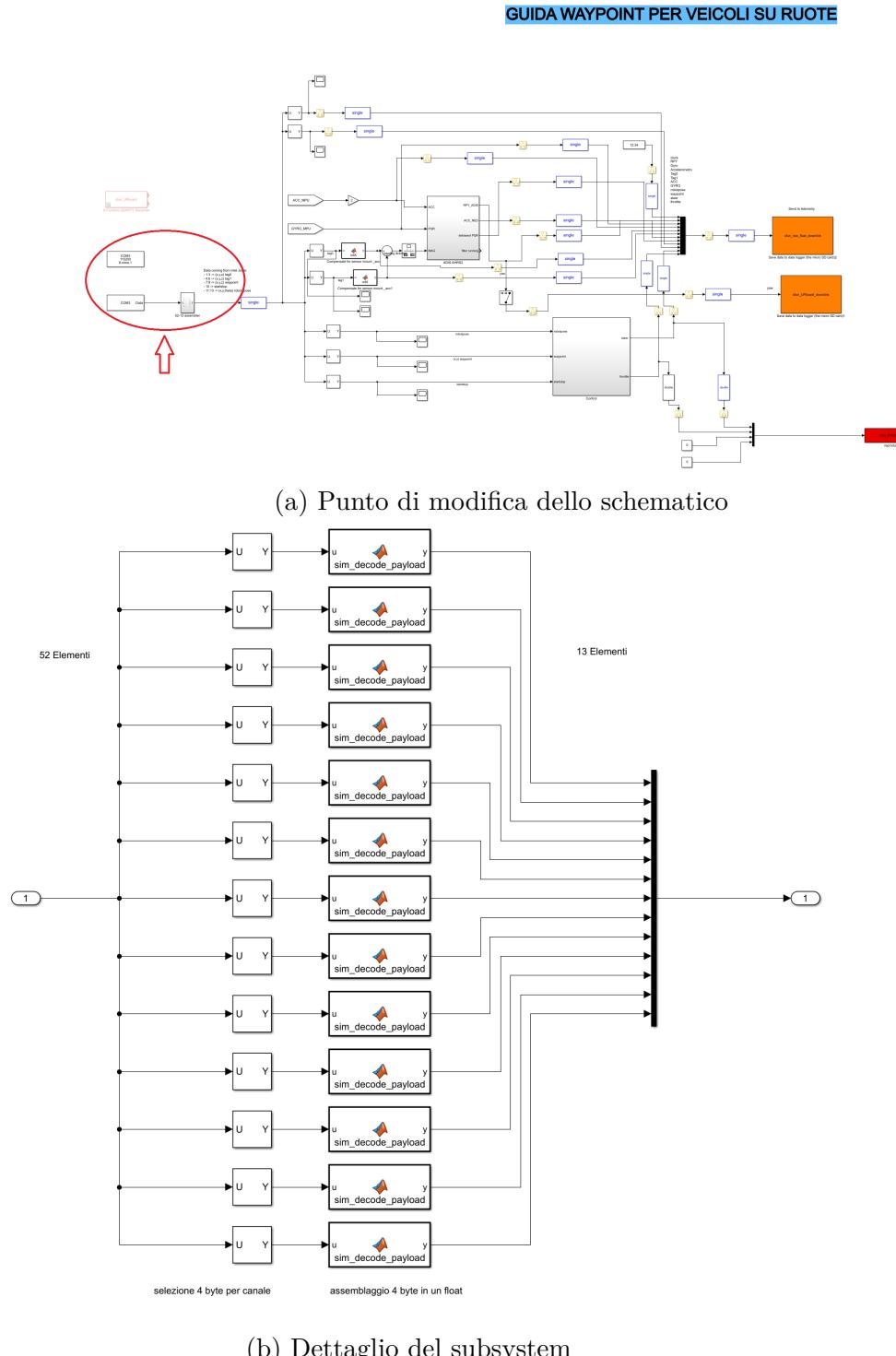
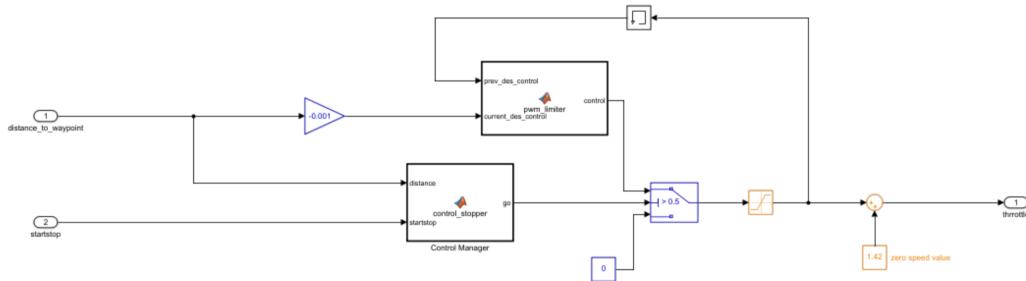


Figura 2.3 Modifica schematico Simulink per connessione Arduino e simulazione su PC

Poiché la causa del problema è stata localizzata con certezza all'interno dell'STM® ma dai test eseguiti non sono state ottenute le risposte desiderate, si è proceduto ad un'indagine ancora più approfondita sullo schematico Simulink® in esecuzione sull'STM®, andando a ispezionare anche i livelli in cui viene configurato l'hardware. Si è quindi scoperto che il modulo GPS risultava abilitato nonostante non fosse presente e che eseguiva frequenti letture sulla medesima porta seriale utilizzata dal controllo, corrompendone i dati. La disattivazione del modulo GPS ha risolto il problema definitivamente, aprendo la strada a test e sperimentazioni.

Corretti i problemi di comunicazione, è stato possibile analizzare il controllo implementato, che si è rivelato migliorabile: il sistema di controllo presente per l'acceleratore presentava più blocchi e in uscita forniva il tempo alto del segnale PWM di controllo semplicemente ponendolo uguale al segnale di errore (in metri) e limitandolo con una saturazione.



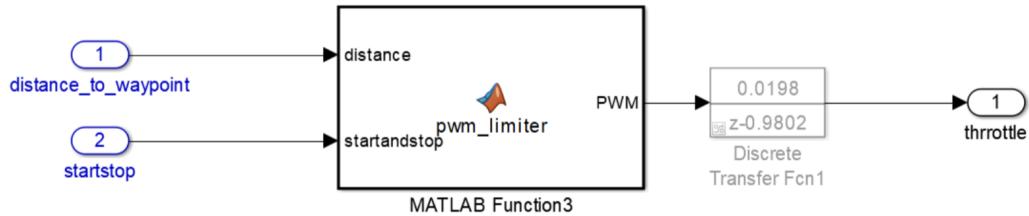


Figura 2.5 Simulink - Controllo finale

## 2.3 Localizzazione robusta

Per rendere robusta la localizzazione tramite il Lidar, è stata implementato un sistema ausiliario che invia dei fix di posa quando la stima fornita da AMCL differisce eccessivamente dalla posizione fornita dal sistema UWB.

Ciò può accadere principalmente nei seguenti scenari:

- Fallimento dello scan matching, cioè quando non si riesce ad associare correttamente le scansioni ricevute alla mappa e di conseguenza la posa inferita da AMCL è errata;
- “Kidnapped robot problem”, cioè quando il robot viene forzatamente spostato in una nuova posizione arbitraria oscurando il sensore Lidar in modo che durante lo spostamento non sia in grado di cogliere variazioni nello scan e di conseguenza aggiornare la sua posizione. Quando poi il sensore viene scoperto e la scansione può ripartire, AMCL continuerà a dare come posa stimata quella precedente allo spostamento, nonostante non ci sia più match con la mappa.

Questi problemi sono stati risolti sviluppando il nodo `pose_fix_pub` il quale osserva costantemente sia i dati di posa del punto medio tra le tag, sia la stima fornita da AMCL; istante per istante calcola l'errore (in norma 2) tra le due, e qualora questo risulti superiore ad 1.5m, avvia la procedura di fixing di posa descritta di seguito:

1. Salvataggio della posa del punto medio tra le tag, corrispondente all'incirca alla posizione del Lidar sul robot, con orientazione letta dall'STM®;
2. Disabilitazione dei motori per motivi di sicurezza - funzione attualmente non attiva
3. Pubblicazione della posa appena salvata sul topic `/initialpose` a cui è iscritto AMCL e che ha come effetto una reinizializzazione del filtro nel punto indicato

con covarianza pari a quella che si avrebbe all'avvio. A questo punto, il filtro attenderebbe uno spostamento o una rotazione per effettuare un update della stima di posa. L'entità delle variazioni necessarie è specificata nei parametri `update_min_d` (minima traslazione) e `update_min_a` (minima rotazione);

4. Per forzare il filtro a portare a convergenza la stima riallineandosi con la mappa mentre il robot è fermo, viene effettuata una modifica online del parametro `update_min_d` di AMCL settandolo a zero tramite chiamata al service `dynamic_reconfigure` che reinizializza il filtro con il parametro scelto;
5. Attesa, necessaria alla convergenza del filtro;
6. Nuova chiamata al service `dynamic_reconfigure` per riportare il parametro `update_min_d` al valore precedente (questo causa una nuova reinizializzazione del filtro)
7. Riabilitazione dei motori(se prima erano abilitati) - funzione attualmente non attiva

Lo schema dell'algoritmo è mostrato nella Figura 2.6.

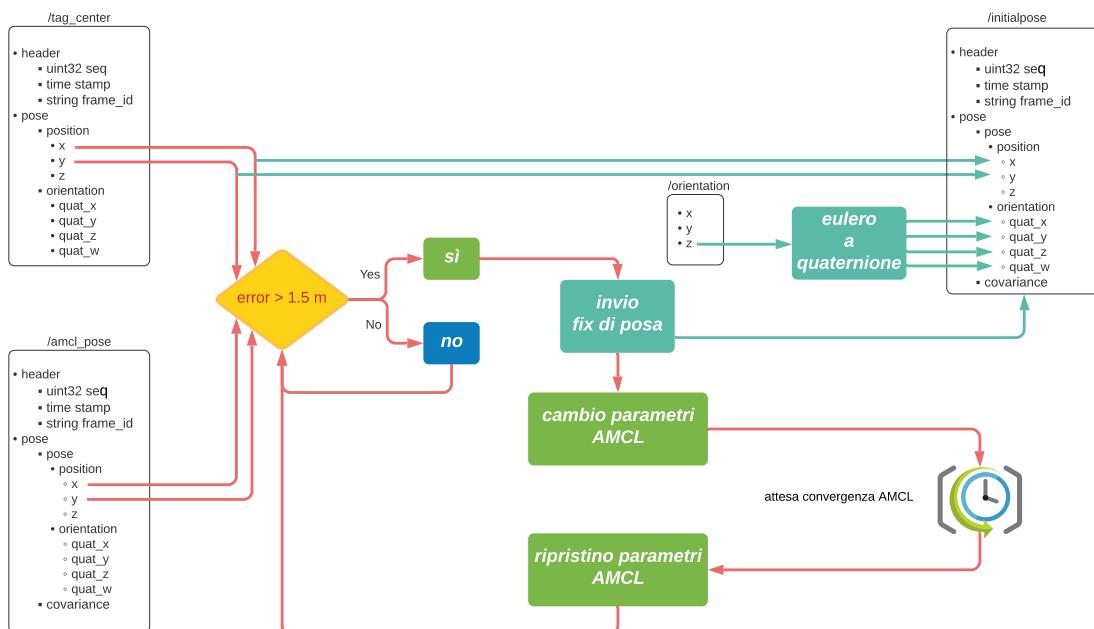


Figura 2.6 Fix di posa

Il tempo totale per completare una reinizializzazione a seguito di un fix di posa è influenzato da vari fattori: vi è un ritardo fisso di 4s impostato via software per motivi di stabilità<sup>4</sup>, a cui si sommano circa 10-15s necessari ad AMCL per gestire le due reinizializzazioni e convergere (la variabilità è legata alla "bontà" della stima ricevuta, a fix particolarmente buoni corrisponderà un tempo di convergenza ridotto, mentre fix peggiori richiederanno più passi dell'algoritmo per convergere). Nonostante i molteplici tentativi, non è stato possibile ridurre questi tempi di attesa.

### 2.3.1 Oscuramento Lidar

Quando il Lidar è oscurato, il filtro a particelle (AMCL) non è in grado di fornire alcun update per la stima di posa del robot e di conseguenza non vengono pubblicati messaggi sul topic `robot_pose` che deriva da `amcl_pose` e necessario per il controllo dei motori. A livello di interfaccia grafica, ciò corrisponde a vedere invariata la posa del robot su Rviz anche mentre quest'ultimo viene spostato rigidamente. Senza ulteriori accorgimenti, finché il Lidar non tornerà online, per via del funzionamento di AMCL, non sarà possibile aggiornare la posa del robot che pertanto rimarrà pari all'ultima disponibile prima dell'oscuramento del sensore. Durante tutto questo intervallo non si avrà alcuna informazione su eventuali spostamenti del robot.

Tornato online il Lidar, AMCL riprenderà a pubblicare ed arriverà anche un fix di posa a causa della discordanza con le UWB e solo a conclusione di questa fase sarà nuovamente disponibile una posa affidabile.

Per avere sempre una traccia degli spostamenti del robot anche nella situazione sopra descritta, si è ideato un procedimento che, osservando i dati provenienti dal Lidar, è in grado di sostituirsi ad AMCL come fonte di dati per `robot_pose`.

L'elemento chiave è il campo `intensities` del messaggio<sup>5</sup> che viene trasmesso sul topic `scan`, il quale è costituito da un array che contiene valori non nulli solo quando il Lidar funziona correttamente. Pertanto, monitorandolo continuamente si è in grado di stabilire quando si hanno malfunzionamenti ed intervenire, sfruttando i dati provenienti dalle UWB come sorgente per `robot_pose`. Più in dettaglio, si fa riferimento ai messaggi sul topic `tag_center`, contenenti la posa del punto medio tra le due tag, con orientazione ottenuta dall'STM, i quali sono già riferiti al frame `map` e quindi coerenti con quelli si avrebbero da AMCL.

---

<sup>4</sup>La procedura di reinizializzazione online di AMCL non ha sempre esito positivo causando talvolta la morte del processo; la frequenza con cui queste reinizializzazioni sono richieste è determinante, richieste troppo ravvicinate hanno sempre condotto a crash di sistema.

<sup>5</sup>Di tipo `LaserScan`

Uno schema di funzionamento di tale sistema è descritto in Figura 2.7.

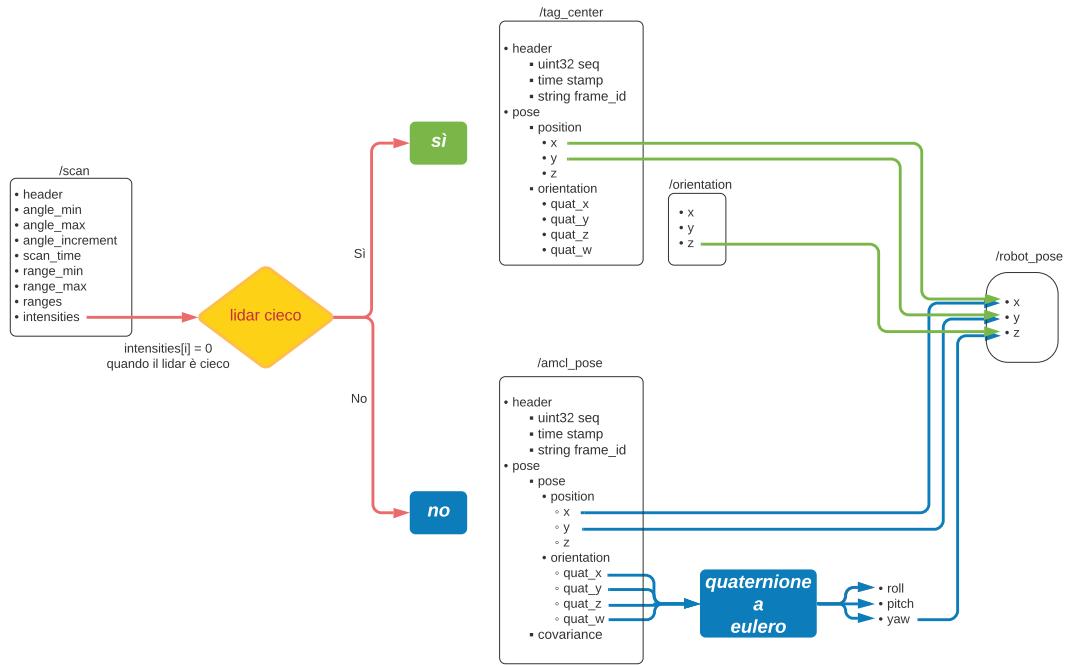


Figura 2.7 Schema `amcl2robotpose`



# Capitolo 3

## Dati e esperimenti

### 3.1 Esperimento 1 - Fix di posa simulato con Rviz

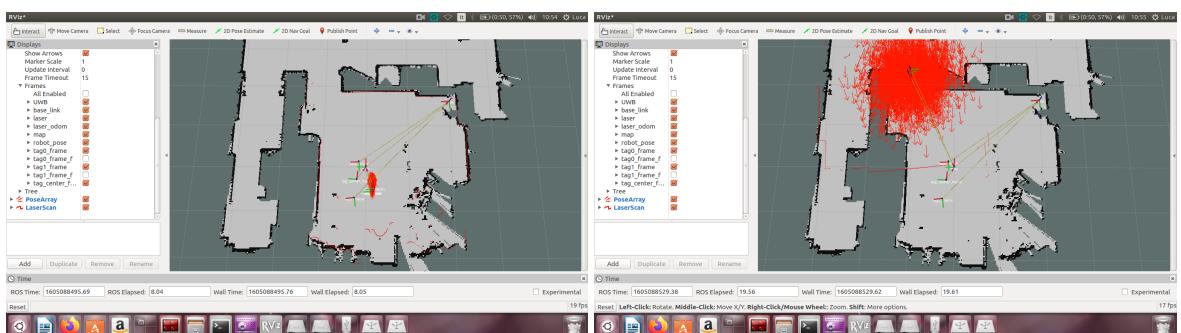
Nel primo esperimento l'obiettivo è quello di dimostrare la bontà della procedura di fix di posa messa a punto.

Per fare ciò, dopo che l'algoritmo AMCL si è avviato ed il robot si è correttamente localizzato all'interno dell'ambiente (Figura 3.1a), si è simulata l'eventualità che il veicolo si perda.

Ci si aspetta che la procedura ideata sia in grado di identificare tale problema e successivamente fornire la posa corretta su cui reinizializzare il filtro.

Il fenomeno è stato simulato impostando una posa sbagliata tramite il comando `2DPoseEstimate` presente su Rviz, che permette di inviare una stima di posa su cui il filtro a particelle andrà poi a reinizializzarsi (Figura 3.1b).

Come previsto, l'algoritmo di controllo provvede subito a fornire una correzione, riportando la stima nel punto in cui il robot si trova veramente (Figura 3.1c) e facendo poi convergere il filtro (Figura 3.1d).



(a) Situazione di partenza

(b) Pose fix forzato errato

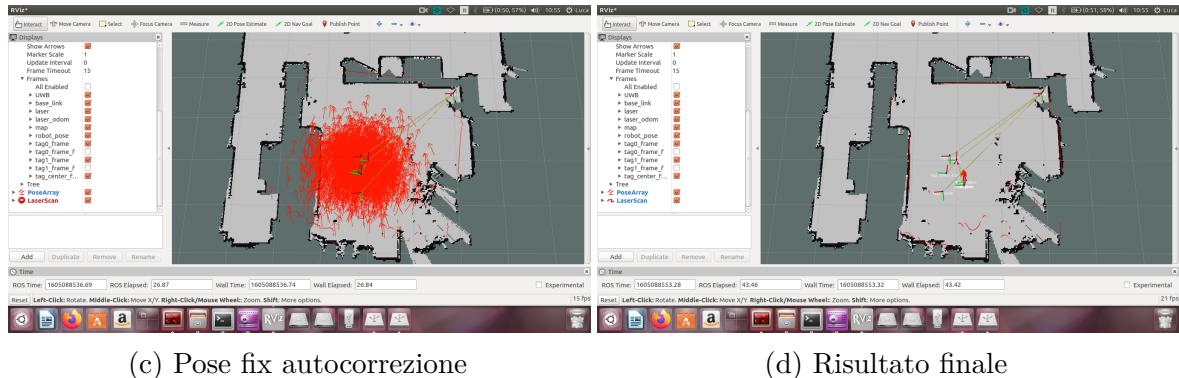


Figura 3.1 Esperimento 1

## 3.2 Esperimento 2 - Fix di posa durante navigazione in più stanze

Nel secondo esperimento ci si è posti nelle condizioni di funzionamento standard, con il robot che naviga tra più stanze, per verificare come si comporta l'algoritmo di localizzazione con i relativi meccanismi di correzione.

La procedura funziona bene e senza problemi finché il robot si trova nella stanza in cui sono presenti le ancore UWB, il veicolo naviga senza problemi e la localizzazione, che avviene tramite AMCL, è sempre molto precisa (Figure 3.2a, 3.2b, 3.2c, 3.2d).

Ciò non succede quando tale stanza viene lasciata, infatti spostandosi in altri ambienti le tag UWB risentono di forti disturbi e questo si traduce in una discordanza tra la posa stimata da AMCL e quella stimata dalle tag, che iniziano ad inviare continui fix di posizione errati (Figure 3.2e, 3.2f).

Questo porta l'algoritmo AMCL a reinizializzarsi di continuo in punti sempre diversi e quasi mai corretti, generando un malfunzionamento di tutta la procedura di localizzazione, che in alcuni casi finisce addirittura per fallire completamente.

Riportando il veicolo nella stanza coperta dalle ancore, dove il segnale UWB risulta molto più stabile e corretto, il tutto riprende a funzionare correttamente (Figure 3.2g, 3.2h).

### 3.2 Esperimento 2 - Fix di posa durante navigazione in più stanze

29

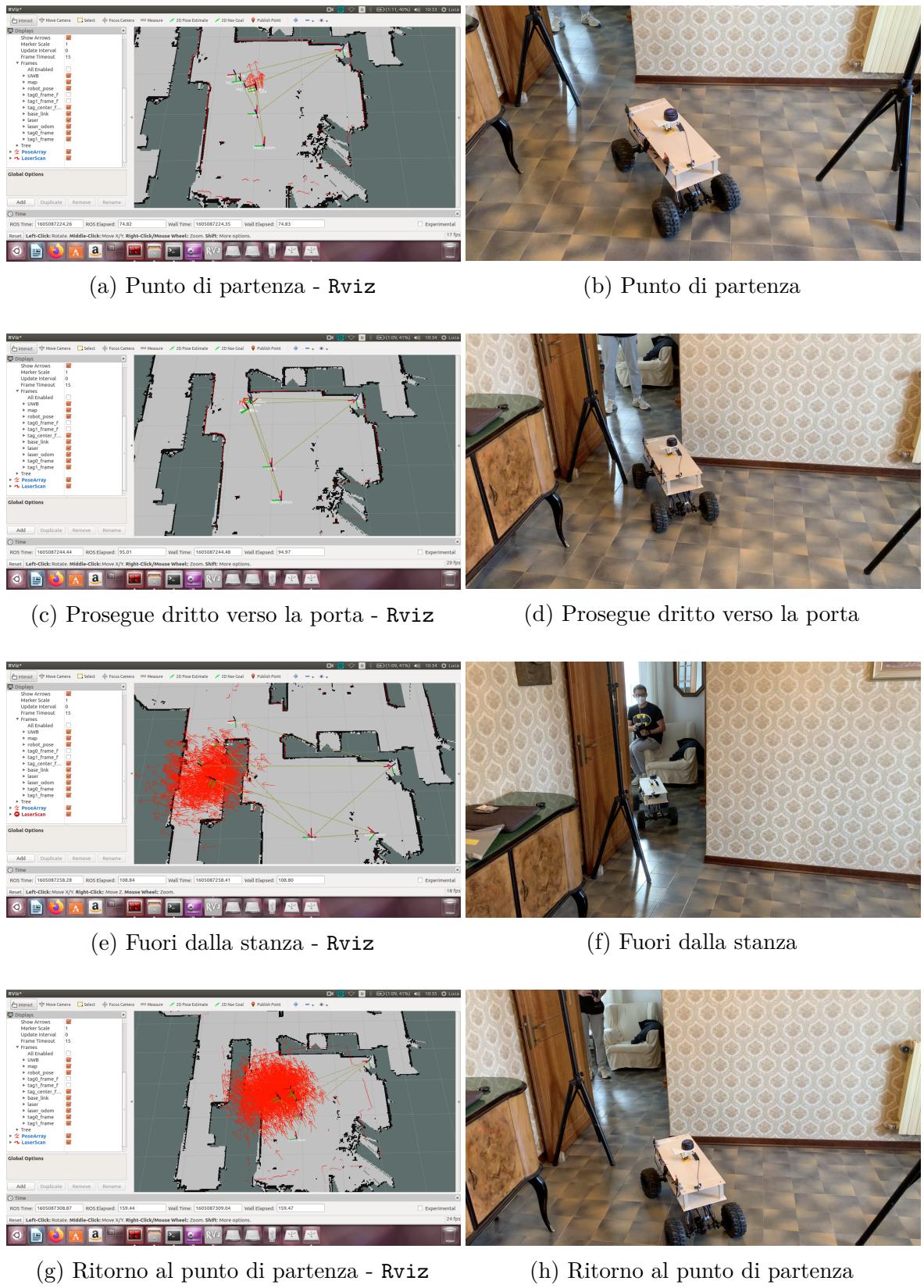


Figura 3.2 Esperimento 2

### 3.3 Esperimento 3 - Rapimento del robot

Nel terzo esperimento si è simulata l'eventualità in cui il Lidar potrebbe non riuscire a vedere oppure sia soggetto ad un malfunzionamento. In questo caso l'algoritmo AMCL, non ricevendo scansioni valide, si arresta e smette di inviare le stime di posa del robot. Senza nessun tipo di accorgimento, questo porterebbe la localizzazione ed il relativo controllo a fallire.

Ecco allora subentrare la navigazione puramente basata sul segnale UWB, che, seppur rumoroso e soggetto ad errori, continua a dare una stima della posizione del robot, che si sostituisce temporaneamente a quella che dovrebbe arrivare dal Lidar, permettendo di continuare l'esecuzione del task assegnato.

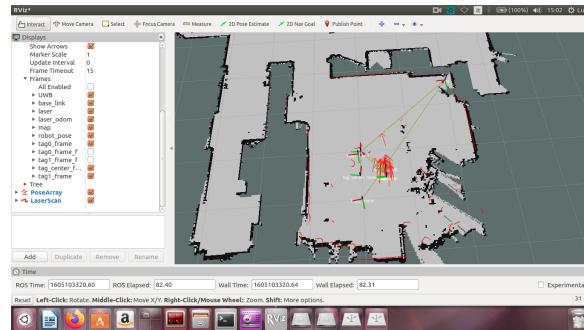
Nel caso in cui il Lidar torni in linea, tutta la procedura di localizzazione passa di nuovo al funzionamento standard, ovvero Lidar come sensore principale e tag sfruttate solo per correzioni di posa in caso di errori.

Per eseguire questo test si è coperto il sensore Lidar, in modo che le scansioni inviate risultassero nulle e poi si è spostato il robot manualmente in una posizione differente (Figure [3.3b](#), [3.3c](#), [3.3d](#), [3.3e](#)). Così facendo, AMCL oltre a non essere in grado di accorgersi delle variazioni, smette di inviare pose aggiornate.

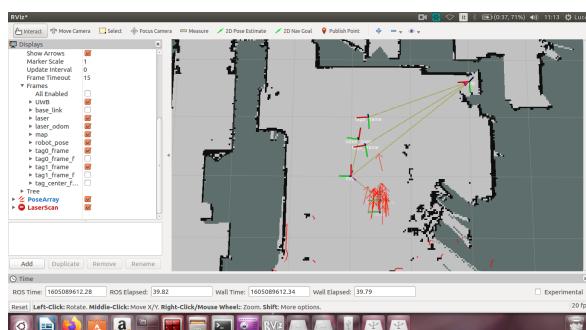
Durante lo spostamento, come anticipato, sono le UWB a mandare il messaggio con la posa del robot aggiornata (Figura [3.3f](#)).

Una volta posizionato in un nuovo punto, differente da quello di partenza, il Lidar è stato nuovamente scoperto. Con il ritorno dei messaggi sul topic `/scan`, è arrivato anche il fix di posa per reinizializzare il filtro a particelle, che come si può apprezzare nelle (Figure [3.3g](#), [3.3h](#), [3.3i](#), [3.3j](#), [3.3k](#)) converge.

A questo punto la navigazione può continuare normalmente.



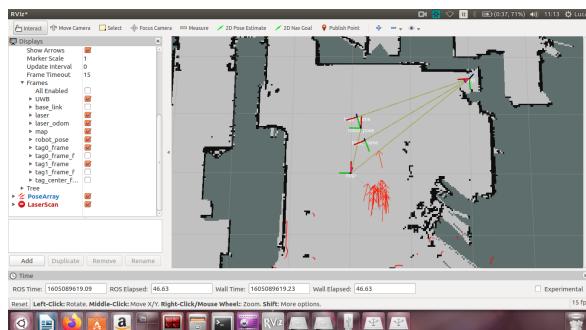
(a) Punto di partenza - Rviz



(b) Rapimento - Rviz



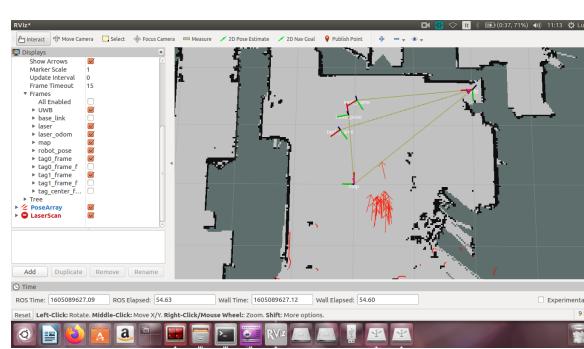
(c) Rapimento



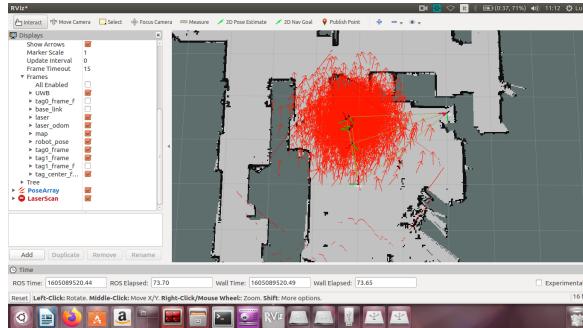
(d) tag\_center sostituisce robot\_pose e si(e) spostamento mantenendo il sensore Lidar vedono i vecchi frame - Rviz



(e) Spostamento mantenendo il sensore Lidar coperto



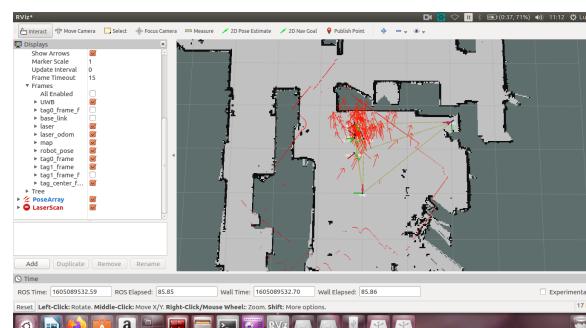
(f) tag\_center sostituisce robot\_pose



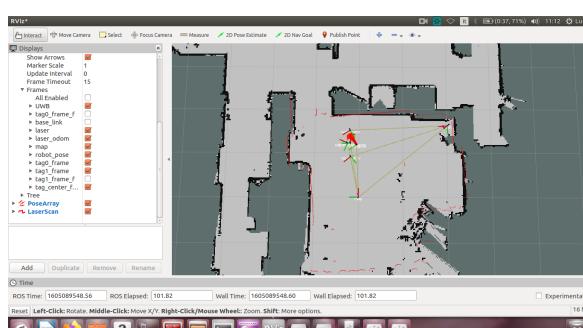
(g) Lidar scoperto e fix - Rviz



(h) Lidar scoperto e fix



(i) Convergenza ed allineamento in corso - Rviz



(j) Nuovamente pronto a navigare - Rviz



(k) Nuovamente pronto a navigare

Figura 3.3 Esperimento 3

### 3.4 Esperimento 4 - Navigazione outdoor

Nel quarto esperimento è stata valutata la capacità di navigazione a waypoint outdoor sfruttando solo i dati provenienti dalle UWB.

Il robot è stato gestito via ssh inviando di volta in volta un goal ed attendendo che vi arrivasse.

Inizialmente il setup prevedeva un posizionamento delle ancore a 20m l'una dall'altra con disposizione a quadrato, ma i risultati ottenuti non erano buoni, il segnale delle tag veniva spesso perduto ed era molto rumoroso, con errori pari anche a più di 3m. Si è quindi passati alla disposizione indicata in (Figura C.1) dove l'asse z è posizionato diversamente; una volta effettuato questo cambiamento i risultati si sono dimostrati buoni, ottenendo quasi sempre un raggiungimento del goal, con il veicolo che si arrestava in un raggio di circa 50cm da esso, coerentemente con l'opzione selezionata a livello di controllo. I motivi dei fallimenti ottenuti sono esaminati di seguito.



Figura 3.4 Posizionamento delle ancore UWB

Al variare della posizione relativa tra goal e punto di partenza, è stato possibile osservare che il controllo implementato sul robot, che non tiene conto in alcun modo della struttura dello stesso, non sempre riesce a definire una traiettoria corretta; infatti, a causa dei vincoli meccanici a cui è soggetto il veicolo, principalmente legati al raggio di sterzata, quando il goal è in una posizione difficilmente raggiungibile (ad esempio un punto interno alla circonferenza definita dal raggio di sterzo) si entra in un loop che

consiste in un movimento circolare attorno ad esso. Viceversa, quando si ha un goal raggiungibile con uno spostamento rettilineo o compatibile con la capacità di sterzo del veicolo, questo non ha alcuna difficoltà a portare a termine il task.

# Capitolo 4

## Conclusioni

A differenza del robot di partenza, vedi Sezione 1.2, il robot funziona.

È in grado di determinare la sua posizione all'interno della mappa, anche senza la stima di posa iniziale<sup>1</sup>, e quando si muove la convergenza è molto rapida. Una volta che si è localizzato può spostarsi liberamente senza perdersi, purché rimanga all'interno di un'area in cui le UWB hanno una buona copertura, altrimenti si verificano i problemi esposti nell'esperimento 2. Nel caso di “kidnapped robot problem” e spostamento manuale del robot, il veicolo è in grado di ritrovarsi non appena il Lidar viene scoperto, con il filtro che converge in maniera soddisfacente, riuscendo ad allineare la mappa senza eccessivi problemi.

Tuttavia quello che emerge dagli esperimenti effettuati, è che la scarsa affidabilità del sistema UWB influenza fortemente i risultati. Infatti, anche se la parte che sfrutta lo scan-matching sta funzionando correttamente, il forte rumore presente sul sistema UWB può causare l'invio dati di posa errati e discordanti che spesso portano al fallimento della localizzazione. In aggiunta a ciò, è necessario sottolineare che la procedura scelta per la reinizializzazione online di AMCL non è sufficientemente robusta, d'altra parte si sta utilizzando il software in un modo non previsto<sup>2</sup>.

Ciononostante, l'obiettivo prefissato di riuscire a navigare all'interno di una mappa nota, combinando le potenzialità dei sensori a disposizione, è stato raggiunto.

---

<sup>1</sup>Alla partenza, la posa iniziale di default di AMCL corrisponde all'origine della mappa e qualora il robot si trovasse ad una distanza superiore ad 1.5m da essa, automaticamente verrebbe inviato un fix di posa. Nel caso in cui la distanza fosse minore, sarà necessario intervenire manualmente.

<sup>2</sup>Nel suo utilizzo standard, AMCL presuppone che il robot abbia sempre una fonte stabile e buona di odometria e che il Lidar non sia mai offline, inoltre richiede uno spostamento non nullo - dell'ordine dei cm - per portare a convergenza la stima.



# **Capitolo 5**

## **Suggerimenti per sviluppi futuri**

- Implementare una GUI semplice in ROS, in modo da rendere l'utilizzo del software sviluppato più user-friendly;
- Indagare sulla possibilità di utilizzare metodi alternativi per ottenere una convergenza più rapida e robusta di AMCL in seguito ad un fix di posa;
- Valutare metodi alternativi per la fusione dei dati in arrivo dai sensori a disposizione;
- Proseguire le indagini sugli effetti del rumore elettromagnetico sul sistema UWB;
- Eventuale studio di una piattaforma meccanica stabilizzata per mantenere il sensore Lidar livellato rispetto al terreno;
- Progetto di un controllo più articolato, aggiungendo la possibilità di fare retro-marcia ed obstacle avoidance;



# Bibliografia

- [1] L. Crosato and C. Tesconi, “Studio e caratterizzazione del sistema pozyx.” Progetto di Sistemi di guida e navigazione, 2017-2018.
- [2] PyPozyx, “Pypozyx’s documentation.” <https://pypozyx.readthedocs.io/en/develop/>, 2018.



# Appendice A

## Cenni teorici

### A.1 Adaptive Monte Carlo Localization

La *Monte Carlo Localization* (MCL), nota anche come Particle Filter Localization, è un algoritmo di localizzazione che sfrutta un filtro a particelle per stimare la posizione e l'orientazione di un robot mentre esso si muove e percepisce l'ambiente all'interno di una mappa nota.

La distribuzione degli stati probabili viene rappresentata come una nuvola di particelle nello spazio di stato (o delle configurazioni)<sup>1</sup>. La stima dello stato corrente del robot ottenuta dal filtro a particelle è una funzione di densità di probabilità (distribuita nello spazio di stato), che al tempo  $t$  è rappresentata da un insieme di particelle  $X_t = \{x_t^1, x_t^2, \dots, x_t^M\}$ . Ogni particella descrive uno stato possibile, e di conseguenza, nel caso di un problema di localizzazione, una posa possibile. Tipicamente l'inizializzazione viene fatta con una distribuzione uniforme delle particelle nello spazio di configurazione, dato che, non avendo informazioni sulla posizione del robot, è lecito assumere che questo possa trovarsi in ogni punto con la stessa probabilità. Ogni volta che il robot si muove, grazie ai dati di odometria, le particelle vengono fatte avanzare effettuando una previsione del nuovo stato e queste, all'arrivo di nuove misurazioni, vengono poi ricampionate basandosi su una stima Bayesiana ricorsiva, vale a dire, quanto bene i dati effettivamente rilevati sono correlati allo stato previsto. In definitiva, le particelle dovrebbero convergere verso la posizione reale del robot, ottenendo una nuvola sempre più piccola e concentrata.

---

<sup>1</sup>Lo spazio delle configurazioni è lo spazio a cui appartiene lo stato del robot, esso dipende dalla specifica applicazione e progettazione. Per esempio, lo stato di un robot capace del semplice moto nello spazio 2D consiste in  $(x, y)$  per la posizione e  $(\theta)$  per l'orientazione.

Le regioni dello spazio di stato dove si addensano più particelle definiscono gli stati più probabili del robot, mentre vale il contrario per quelle con poche particelle. L'algoritmo assume la proprietà di Markov, per cui la distribuzione di probabilità dello stato attuale dipende solo dallo stato precedente, cioè  $X_t$  dipende solo da  $X_{t-1}$ . Questo funziona solo se l'ambiente è statico e non cambia con il tempo.

Elemento fondamentale per il funzionamento dell'algoritmo è la conoscenza della mappa dell'ambiente, l'obiettivo è quindi quello di stimare la posa del robot all'interno di essa. Ad ogni istante  $t$  viene presa come input la stima precedente  $X_{t-1} = \{x_{t-1}^1, x_{t-1}^2, \dots, x_{t-1}^M\}$ , un controllo  $u_t$  e i dati ricevuti dai sensori  $z_t$ ; tramite la funzione `Motion_Update()` ogni particella viene fatta avanzare di un passo temporale (predizione), noti il valore all'istante precedente ed il controllo attuato, successivamente, la funzione `Sensor_Update()` gli associa un peso valutando la correlazione con i dati ricevuti dai sensori.

Segue quindi un ricampionamento delle particelle in base ai pesi appena calcolati; infine si restituisce in output la nuova stima dello stato (nuvola)  $X_t$ . I passi dell'algoritmo sono descritti di seguito.

---

**Algorithm 1:** Algoritmo MCL

---

```

input :  $X_{t-1}, u_t, z_t$ 
output :  $X_t$ 

1  $\bar{X}_t = X_t = 0$ ;
2 for  $m = 1$  to  $M$  do
3    $x_t^{[m]} = \text{Motion\_Update}(u_t, x_{t-1}^{[m]})$ ;
4    $w_t^{[m]} = \text{Sensor\_Update}(z_t, x_t^{[m]})$ ;
5    $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ ;
6 end
7 for  $m = 1$  to  $M$  do
8   | draw  $x_t^{[m]}$  da  $\bar{X}_t$  with probability  $\propto w_t^{[m]}$ ;
9 end
10 return  $X_t$ 

```

---

Inizialmente, è necessario utilizzare un grande numero di particelle ( $M$ ) in modo da coprire l'intera mappa con una distribuzione uniformemente casuale. Tuttavia,

quando le particelle iniziano a convergere intorno ad una certa posizione, mantenere una dimensione del campione così grande è uno spreco computazionale.

L'algoritmo MCL può essere quindi migliorato campionando le particelle in modo adattivo sulla base di una stima di errore utilizzando la divergenza Kullback-Leibler (KLD), è qui che si introduce la *Adaptive Monte Carlo Localization* (AMCL), una variante del MCL semplice. In questo algoritmo, ad ogni iterazione viene calcolata una nuova dimensione del campione  $M_x$  (numero di particelle) in modo tale che l'errore tra l'approssimazione a posteriori<sup>2</sup> e quella sample-based<sup>3</sup> sia minore di un certo  $\epsilon$  con probabilità  $1 - \delta$ , dove  $\delta$  ed  $\epsilon$  sono parametri fissati.

In estrema sintesi, l'algoritmo AMCL, avvicinandosi alla convergenza, riduce progressivamente il numero di particelle utilizzate riducendo il carico computazionale.

Più in dettaglio, AMCL gestisce in modo adattivo il numero di particelle  $M_x$ : l'idea principale è quella di creare una griglia (istogramma) sovrapposta allo spazio di stato dove ogni cella dell'istogramma è inizialmente vuota; ad ogni iterazione, una nuova particella viene ricavata dall'insieme di particelle precedenti e ricadrà di conseguenza in una determinata cella. L'algoritmo controlla quali celle vengono riempite e tiene traccia del loro numero -  $k$  - e se una particella viene inserita in una cella precedentemente vuota, il valore di  $M_x$  viene ricalcolato (questo aumenta circa in modo lineare in  $k$ ). Il ricampionamento viene ripetuto fino a quando la dimensione corrente  $M$  del campione è pari ad  $M_x$ , che sarà quasi sempre minore del numero massimo di particelle possibili.

Di fatto, quando il ricampionamento porta ad un progressivo addensarsi delle particelle che quindi ricadranno in un numero  $k$  sempre più ridotto di celle, si ridurrà anche il numero complessivo di particelle  $M_x$ . Viceversa, nel caso in cui il ricampionamento non porti le particelle a concentrarsi (ad esempio quando si hanno misure e predizioni discordanti), bensì a spargersi, il numero di celle occupate tenderà ad aumentare e questo determinerà anche un aumento del numero di future particelle considerate. In conclusione AMCL ha come caratteristica chiave un numero di particelle adattivo in base al corrente stato di convergenza della stima: più la stima è concentrata (bassa covarianza) e meno particelle verranno usate per il suo tracking, mentre più la stima è incerta e più saranno le particelle utilizzate.

---

<sup>2</sup>La stima dello stato basata sulla stima all'istante precedente propagata con i dati forniti dall'odometria.

<sup>3</sup>La stima dello stato basata sullo scan matching.



# Appendice B

## Guida al Codice

In questa Appendice si descrive il codice utilizzato suddividendo in package, la cui repository è disponibile all'indirizzo [GitHub](#).

All'utilizzo pratico, i file qui descritti vengono eseguiti tramite 5 launch file presenti all'interno del package `charlie_pkg` realizzato ad hoc per il robot.

I 5 launch file sono:

- `save_map_origin.launch` – Salva la tf fra il sistema di riferimento delle UWB e il sistema di riferimento della mappa. Vengono lanciati i nodi `Serial_Manager_Luca` per leggere l'heading dall'STM® e `save_uwb2map_TF.py` (appartenente al package `pozyx_ros`)<sup>1</sup>;
- `new_map.launch` – Avvia l'acquisizione di una nuova mappa mostrando il processo su `Rviz`<sup>2</sup>. Prima di effettuare una nuova scansione salvare sempre il punto di partenza (che diverrà l'origine della nuova mappa) tramite `save_map_origin.launch`;
- `start_uwb.launch` – Avvia la recezione dei dati dalle tag del sistema UWB. I nodi lanciati sono `pose_pub_2_tag.py`, `filter_stamped.py` e `filter_stamped_second_ch.py` (package `pozyx_ros`);
- `test_uwb_map.launch` – Avvia i nodi che permettono di vedere le pose dei tag nella mappa. Dopo aver caricato la mappa, avvia il nodo `tag_in_map.py` il quale, leggendo i dati salvati in precedenza<sup>3</sup>, pubblica la tf tra il frame UWB e quello della mappa. Oltre a ciò viene pubblicata la posa del punto centrale tra le due

---

<sup>1</sup>I dati vengono salvati nel file `UWB2map_TF.txt`

<sup>2</sup>Si ricorda che, una volta raggiunto un risultato soddisfacente, sarà necessario salvare la mappa tramite il comando `rosrun map_server map_saver -f <nome_mappa>`

<sup>3</sup>Legge il file `UWB2map_TF.txt`

tag sia come tf che su topic, prendendo come orientazione quella resa disponibile dal filtro implementato sull'STM®.

- `localization.launch` – Launch file che avvia il sistema di localizzazione AMCL + UWB ed i nodi necessari al suo funzionamento.

## **pozyx\_ros**

- `defs.py`
- `pose_pub_1_tag.py`
- `pose_pub_2_tag.py`
- `filter_stamped.py`
- `filter_stamped_second_ch.py`
- `save_uwb2map_TF.py`
- `tag_in_map.py`
- `pose_fix_pub.py`
- `pose_to_odom.py`
- `amcl2robot_pose.py`

"`defs.py`" contiene tutte le definizioni dei nomi dei frame e dei topic utilizzati dai nodi che sfruttano il sistema UWB. Questo file è incluso in tutti gli script presenti nel package `pozyx_ros` così da avere un riferimento comune per i nomi.

"`pose_pub_1_tag.py`" inizializza e legge i dati di una tag sola e li pubblica su un topic dedicato `/tag0_pose` oltre a fornire una `tf` dall'origine del sistema UWB ("UWB") alla tag ("`tag0_frame`"). I dati sono letti alla massima frequenza possibile che da test pratici risulta essere circa 50 Hz e vengono scritti a schermo.

"`pose_pub_2_tag.py`" avvia il nodo responsabile della lettura e pubblicazione dei dati delle tag.

Nella fase iniziale dello script vengono cercate due tag connesse via USB: se non vengono individuate, si ha un errore ed una conseguente richiesta di connettere un

dispositivo Pozyx®; viceversa, se tutto va a buon fine, le tag vengono inizializzate e si ha una verifica tramite ID per controllare che la tag identificata come testa sia quella corretta. Nel caso in cui risultino invertite, lo script procede automaticamente a scambiarle; ciò è necessario perché ad ogni riavvio del robot le porte seriali vengono reinizializzate casualmente da Linux, se quindi si facesse riferimento al solo nome della porta, non vi è garanzia che il dispositivo ad essa connesso rimanga il medesimo tra un avvio e l'altro e questo potrebbe portare ad interpretare i dati della tag in testa come riferiti alla tag in coda e viceversa.

A questo punto l'inizializzazione è conclusa, si procede quindi ad una lettura ciclica dei dati di posa da ogni tag; per ogni nuovo dato viene inviato un messaggio (di tipo `PoseStamped`) sul relativo topic `/tag0_pose` o `/tag1_pose` e vengono stampati a schermo i valori letti. Parallelamente all'invio dei messaggi, si ha anche la pubblicazione di due `tf`<sup>4</sup>, una per ogni tag - da UWB a `tag0_frame` e `tag1_frame` - che descrivono la trasformazione dal frame UWB al frame centrato sulla tag.

"`filter_stamped.py`" avvia il nodo che si occupa di filtrare i dati di posa di tag0 applicando una media mobile con finestra di ampiezza 10 campioni. All'arrivo di ogni nuova posa la finestra di campioni viene avanzata e viene calcolata la media semplice, pubblicata sul topic `/tag0_pose_f` parallelamente ad una `tf`<sup>4</sup> da UWB a `tag0_frame_f`.

"`filter_stamped_second_ch.py`" avvia il nodo che si occupa di filtrare i dati di posa di tag1; è un codice analogo a quello appena descritto per la prima tag, differisce solo per i topic su cui pubblica che in questo caso fanno riferimento alla tag1.

"`save_uwb2map_TF.py`" avvia il nodo responsabile del salvataggio della posa relativa tra l'origine della mappa (il punto in cui si inizia l'acquisizione) e l'origine del sistema UWB. Il nodo si iscrive ai due topic delle tag (`/tag0_pose` e `/tag1_pose`) ed al topic contenente l'orientazione letta dall'STM® (`/orientation`), dopodiché procede ad osservare i dati raccogliendo un numero configurabile di campioni<sup>5</sup> che vengono prima filtrati per eliminare eventuali outliers (Figura B.3) e quindi mediati per ripulire le pose delle due tag dal rumore. Viene preso il punto medio tra le due e si considera come orientazione (*yaw*) quella letta dal topic `/orientation`. È importante notare che, affinché la trasformazione tra sistema di riferimento UWB e sistema di riferimento mappa sia memorizzata correttamente, è fondamentale che il filtro presente a bordo dell'STM<sup>6</sup> sia arrivato a convergenza: il tempo necessario, con una stima dall'alto, è

---

<sup>4</sup>le `tf` pubblicate, per quanto riguarda la rotazione, tengono conto solo di quella sull'asse *z*

<sup>5</sup>correntemente pari a 100

<sup>6</sup>Si ricorda che il valore dello yaw è letto appunto dall'STM

circa 30 secondi, misurato sperimentalmente osservando l'andamento dei dati in Figura B.1a. È stato osservato l'andamento dell'uscita del filtro prima con ingresso nullo, cioè con il sistema UWB non attivo, attendendo l'arrivo del regime; dopodiché è stato attivato il sistema UWB con conseguente invio della posizione delle tag all'STM® ed è stato nuovamente osservato il transitorio del valore dello yaw<sup>7</sup>. Analizzando il grafico è possibile individuare 4 istanti fondamentali:

- punto 1 - istante in cui l'STM® inizia a trasmettere lo yaw, con il sistema UWB non ancora acceso il filtro si porta verso il valore di riferimento per lo yaw;
- punto 2 - istante in cui il filtro è arrivato a convergenza in assenza di segnale UWB, quindi con ingresso ancora nullo;
- punto 3 - istante di accensione del sistema UWB e conseguente trasmissione delle coordinate delle tag all'STM®, il filtro si porta verso il valore corrispondente all'angolo che la congiungente delle due tag forma con l'asse x del sistema UWB;
- punto 4 - istante in cui si assume il filtro arrivato a convergenza con in ingresso i dati delle UWB.

Osservando in dettaglio il tempo trascorso tra gli istanti 1 e 2 (Figura B.1b) è possibile stimare il tempo di settling in circa 30 secondi, ed anche dal focus sugli istanti 3 e 4 (Figura B.1c) è lecito dedurre un tempo simile. Per una questione di affidabilità è stato inserito un tempo di attesa di 35 secondi per assicurare il raggiungimento della convergenza prima del salvataggio della trasformazione; tuttavia, questo è necessario solo se si desidera avviare questo nodo subito dopo l'accensione del robot, altrimenti, in uno scenario più probabile, in cui si inizia ad operare con il robot già acceso ed i nodi di trasmissione seriale e ricezione da UWB avviati da qualche minuto, non sarebbe necessaria alcuna attesa. Lo stato del salvataggio è reso noto all'utente tramite un messaggio stampato a schermo, osservabile in Figura B.2a ed in Figura B.2b in cui la procedura è giunta al termine.

La posa appena ottenuta viene salvata nel file /home/robot/charlie\_ws/src/pozyx\_ros/src/scripts/my\_pozyx/UWB2map\_TF.txt sotto forma di punto+quaternione, dopodiché il nodo termina automaticamente.

---

<sup>7</sup>Si ricorda che lo yaw è misurato in un sistema di riferimento z-down.

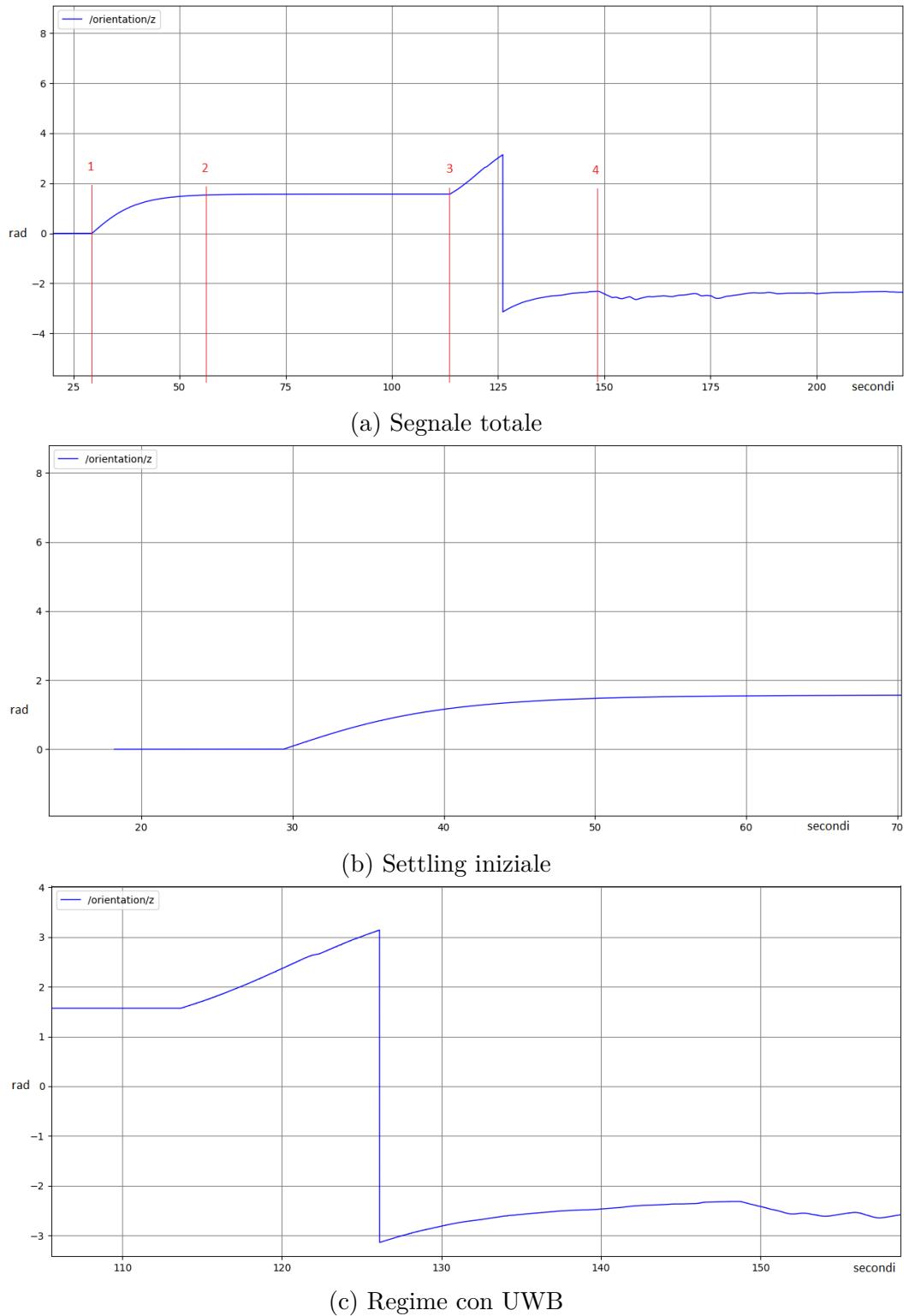


Figura B.1 Grafico dell'analisi del tempo di regime dello yaw letto da STM

Per rimuovere gli outliers si è deciso di basare l'identificazione sullo **Z-score**.

Lo **Z-score** è una misura che descrive la relazione di un valore rispetto alla media di un gruppo, più precisamente è una misura in termini di deviazione standard dalla media, che può essere positivo se sopra di essa o negativo se al di sotto. I dati con valori di **Z-score** più alti rispetto ad una determinata soglia saranno considerati come outliers, nella maggior parte dei casi tale soglia è pari a 3.

Per calcolarlo verrà usata la funzione "`stats.zscore()`" facente parte della libreria `scipy`

Per prima cosa si calcola la norma due al quadrato di ogni dato raccolto:

```

1  for i in range (len(pose_rec_0)):
2      data.append( pose_rec_0[i].pose.position.x**2 +
3          pose_rec_0[i].pose.position.y**2 + pose_rec_0[i].pose.position.z**2 )

```

Poi si crea un `DataFrame`, una struttura dati di Pandas<sup>8</sup>, che andrà a contenere le norme appena calcolate ed infine si riempie una colonna di tale struttura con i valori dello **Z-score** associati ad ogni dato. A questo punto si riempie l'array `indexes` con gli indirizzi dei valori che hanno uno **Z-score** superiore a 3 e quindi classificati come outliers, che poi verranno sfruttati per sapere quali dati andare a rimuovere.

```

1 import pandas as pd
2 from scipy import stats
3 import numpy as np
4
5 z_limit = 3
6
7 data=[]
8 # calcola la norma due al quadrato di ogni dato
9 for i in range (len(pose_rec_0)):
10     data.append(pose_rec_0[i].pose.position.x**2 +
11         pose_rec_0[i].pose.position.y**2 + pose_rec_0[i].pose.position.z**2)
12
13 # identifica gli outlier basandosi sulla norma due al quadrato e
14 # salva il loro indice
15 df= pd.DataFrame({'data':data})
16 df['z_score']=stats.zscore(df['data'])
17 indexes = df.loc[df['z_score'].abs()>=z_limit].index

```

---

<sup>8</sup>libreria software per Python utilizzata per manipolazione e analisi dei dati.

```

18
19 newdata = np.squeeze(df.data)
20 plt.plot(newdata)
21
22 # rimuovi gli elementi in base agli indici degli outlier
23 for i in indexes:
24     del pose_rec_0[i]
25     del data[i]
26
27 newdata = np.squeeze(data)
28 plt.plot(data)
29 plt.show()
30
31 rospy.loginfo('outliers eliminati in tag0: %d',
32 rcvd_msgs_0-len(pose_rec_0))

```

```

        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
data collection terminated...
        TAG_0: 100 / 100 samples collected
        TAG_1: 100 / 100 samples collected
evaluating TF: UWB -> odom
waiting for yaw to settle ... (35s)

```

(a) Aspettando lo yaw

```

collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
collecting data...
        TAG_0: 99 / 100 samples collected
        TAG_1: 100 / 100 samples collected
data collection terminated...
        TAG_0: 100 / 100 samples collected
        TAG_1: 100 / 100 samples collected
evaluating TF: UWB -> odom
waiting for yaw to settle ... (35s)
done
Filtro gli outliers nei dati raccolti
outliers eliminati in tag0: 1
outliers eliminati in tag1: 0
Saved TF from UWB to map in /home/robot/charlie_ws/src/pozyx_ros/src/scripts/my_pozyx/UWB2map_TF.txt
Shutting down node...

```

(b) Termine procedura

Figura B.2 Feedback all'utente dal sistema di salvataggio della posa relativa tra mappa e UWB

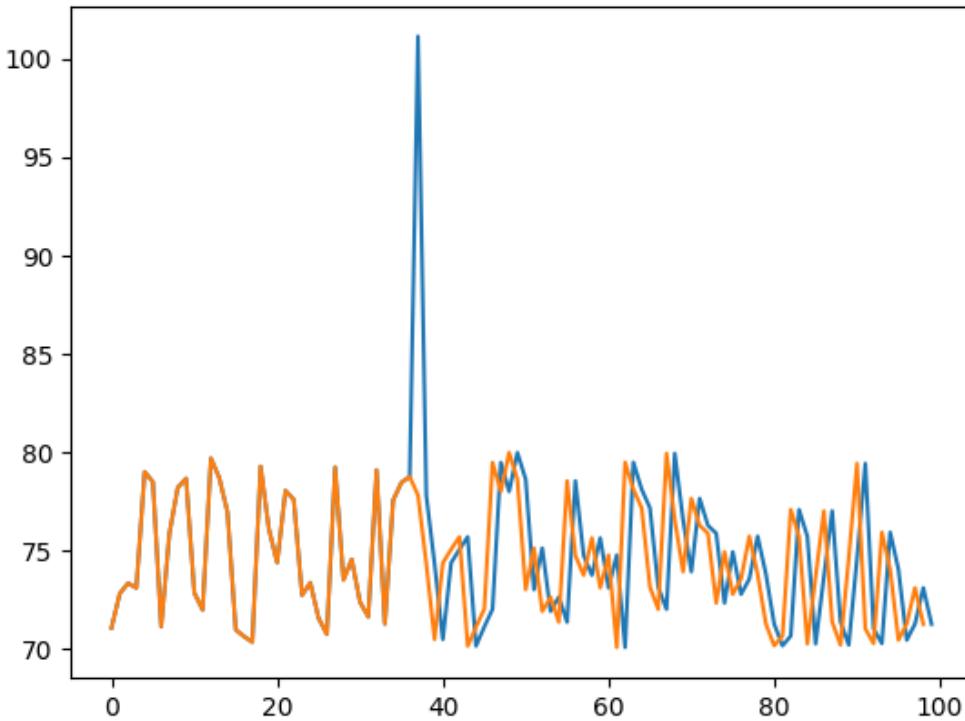


Figura B.3 Rimozione outlier

"`tag_in_map.py`" avvia il nodo che legge dal file `UWB2map_TF.txt` i dati necessari a definire la trasformazione dal frame UWB al frame `map`, utilizzati poi per riportare le coordinate delle tag nel frame della mappa. Pubblica quindi la posa del punto medio tra le due tag, con orientazione letta dal topic `/orientation` proveniente dall'STM®. Per un riscontro grafico, è possibile osservare i dati convertiti tramite `Rviz` grazie a due `tf` pubblicate da questo nodo (`UWB -> map` e `map -> tag_center_frame`), dove il frame corrispondente al punto medio tra le tag assume il nome di `tag_center_frame`.

"`pose_fix_pub.py`" avvia il nodo che, iscrivendosi ai topic di posa di AMCL e del punto medio tra le tag, controlla che le due non differiscano eccessivamente l'una dall'altra; infatti, può capitare che il sistema di localizzazione Lidar fallisca nel caso in cui l'ambiente sia troppo omogeneo oppure in caso di malfunzionamento del sensore (casuale o indotto). In questa eventualità, la probabilità che il sistema converga nuovamente alla posa corretta senza ausili esterni è molto bassa, per ovviare a ciò si è deciso di sfruttare l'informazione fornita dal sistema UWB che è sempre disponibile e non è soggetta a tali problematiche. Qualora le due pose differiscano più di 1.5m viene

automaticamente pubblicato un fix di posa sul topic `/initialpose` al quale segue una modifica temporanea dei parametri di AMCL in modo da forzare la convergenza del filtro senza necessità del movimento del robot. È stato predisposto un meccanismo di protezione che ferma i motori durante la procedura di correzione e che, una volta terminata, ripristina lo stato precedente del controllo. Questo sistema è stato implementato per evitare movimenti del robot mentre la posizione non è nota (questa funzionalità è attualmente disabilitata). Per una descrizione in dettaglio vedere paragrafo [2.3](#).

"`pose_to_odom.py`" avvia il nodo capace di calcolare l'odometria del robot a partire dai dati di una singola tag. Questi sono pubblicati sia come `tf` che come topic. A causa dei risultati non soddisfacenti ottenuti sfruttando questa sorgente di odometria, si sconsiglia l'utilizzo di questo nodo.

"`amcl2robot_pose`" nodo che converte un messaggio di tipo `PoseWithCovarianceStamped` fornito da AMCL in un messaggio di tipo `Point` e lo pubblica sul topic `/robot_pose`.

Effettua inoltre un controllo sul funzionamento del sensore Lidar, infatti quando esso è oscurato o non funziona, il campo `intensities` del messaggio `LaserScan` è riempito con zeri. Se ciò accade la fonte di dati per `/robot_pose` diventa il punto medio delle tag<sup>9</sup> e viene mostrato a schermo che è in corso una navigazione basata su UWB.

## rplidar\_ros

- `rplidarNode`

Avvia il sensore Lidar.

## hector\_slam/hector\_mapping

- `hector_mapping_node`

Nodo standard di `hector_mapping` che si occupa di salvare i dati provenienti del Lidar in una mappa.

---

<sup>9</sup>in coordinate MAP.

## **scan\_tools    kinetic**

- `laser_scan_matcher_node`

Nonostante il nome possa trarre in inganno, questo pacchetto non effettua alcun tipo di scan-match con mappa, bensì confronta scan consecutivi (messaggi di tipo `sensor_msgs/LaserScan`) da cui ricava una stima della posizione del laser che pubblica come `geometry_msgs/Pose2D` oppure come `tf`. Il pacchetto può essere usato senza stime di odometria provenienti da altri sensori, in questo caso è in grado di fornirne lui stesso una stima; questo è il contesto in cui è attualmente utilizzato il nodo. Tuttavia nel caso si disponesse di più fonti di odometria, esse possono essere sfruttate per migliorare i risultati.

## **charlie\_pkg**

- `localization.launch`

"`localization.launch`" è il launch file principale del sistema di localizzazione: in primis avvia il Lidar<sup>10</sup>, carica la mappa e definisce la trasformazione statica da `base_link` a `laser` (il frame solidale al Lidar) posizionandoli come coincidenti. A questo punto l'ambiente di base per la localizzazione è pronto, si avvia quindi `laser_scan_matcher` (nodo `laser_scan_matcher_node`) che si occupa di fornire l'odometria sotto forma di `tf` necessaria ad AMCL per funzionare; vengono poi avviate le due sorgenti di posa del robot, AMCL, che effettua una stima con il sensore laser, e `tag_in_map`, che porta le componenti di posa del punto medio tra le tag in coordinate nel frame `map`. Si hanno a questo punto le due pose del robot riferite allo stesso frame ed è quindi possibile avviare il sistema di fix di posa lanciando il nodo `pose_fix_pub.py`. Da qui, il sistema di localizzazione robusta è in esecuzione e sono quindi disponibili i dati per guidare del veicolo; vengono quindi avviati il nodo gestore della comunicazione seriale ("`SerialManager`") ed i nodi di interfaccia `amcl2robot_pose` e `navgoal` che predispongono la posa stimata da AMCL ed il goal dato tramite `Rviz` nel formato previsto per l'invio in seriale. Il primo si occupa di convertire la posa di tipo `PoseWithCovarianceStamped` fornita da AMCL in Point dove *x* ed *y* sono le coordinate di posizione del robot nella mappa, mentre *z* rappresenta lo *yaw*, questo è necessario per come è stato progettato il sistema di comunicazione via seriale. Il secondo si occupa di effettuare la stessa operazione

---

<sup>10</sup>I parametri sono standard, volendo si può modificare quello riguardante la porta seriale su cui si trova il Lidar (nel nostro caso `/ttyUSB0`)

---

appena descritta, sul goal dato a livello grafico su `Rviz`, cosicchè possa essere inviato via seriale. Per funzionare correttamente deve essere lanciato dopo, o in concomitanza con `start_uwb.launch` per avere accesso ai dati delle UWB.

## Navigation

- `DFL.launch` (per usare le tag come odometria)

"`DFL.launch`" è un launch file gemello di `localization.launch` che, a differenza di quest'ultimo, non utilizza l'odometria ricavata dal laser, bensì la ottiene dal sistema UWB. Dati gli scarsi risultati, questo approccio è stato abbandonato e pertanto non è stata aggiunta alcuna procedura di fix di posa.

## SerialManager

- `SerialManager`

"`SerialManager`" è il nodo che gestisce la comunicazione via seriale, sia in scrittura che lettura. Sfrutta la porta `/dev/ttyUSB1` con BAUD di 115200.

Si iscrive ai topic di posa delle tag e di AMCL<sup>11</sup>, a `/waypoint_publisher` (su cui sono pubblicati  $x$ ,  $y$ ,  $yaw$  della posa desiderata) e `/start_and_stop`(su cui viene pubblicato 0 o 1 per disabilitare o abilitare i motori).

Il codice è abbastanza semplice, si ha una sequenza predefinita di elementi da inviare dove ogni dato viene suddiviso in pacchetti da 4 byte ed inserito nella posizione corretta; una volta costruita la stringa da inviare, questa viene spedita. In ricezione si legge un solo dato `orientation`, proveniente dall'STM® e rappresentante l'heading del veicolo, questo, una volta letto, viene pubblicato sul topic `/orientation`. Per ulteriori dettagli vedere il paragrafo [2.2](#).

---

<sup>11</sup>il topic esatto è `robot_pose`, che contiene le informazioni di  $x$ ,  $y$ ,  $yaw$  prese dalla posa stimata da AMCL



# Appendice C

## Guida all'esperimento

### Avviare interfaccia grafica

Se si desidera collegare l'Intel® Joule™ ad uno schermo tramite cavo HDMI, una volta avviato il robot premere Alt+F2 per attivare la shell ed effettuare il login (nome utente e password sono entrambi: "robot") poi digitare:

```
>> ./init.sh
```

Questo chiederà conferma ("y/n", inserire "y") ed avvierà l'interfaccia grafica.

### Connessione al robot da remoto

Il robot è configurato per generare automaticamente una rete Wi-Fi aperta all'accensione, con nome **rosnet**. Per interagire a distanza, connettersi a questa rete, dopodiché sarà possibile eseguire comandi direttamente sul robot tramite **ssh** come mostrato di seguito.

```
>> ssh -X robot@10.42.0.1 -p 22
```

Alternativamente è possibile utilizzare l'ambiente **ROS** in locale riferendosi comunque al master che è sul robot tramite i comandi (da eseguire su ogni shell che si vuole connettere)

```
>> export ROS_MASTER_URI=http://10.42.0.1:11311  
>> export ROS_HOSTNAME=10.42.0.181
```

In questo contesto non bisogna connettersi tramite `ssh`, tuttavia è necessario essere connessi alla rete `rosnet`): ciò permette ad esempio di sfruttare `Rviz` con le risorse del computer che si sta utilizzando, senza dover fare streaming video dalla Intel® Joule™, metodo molto più lento e poco reattivo.

## Autocalibrazione delle UWB

Le ancore devono essere disposte con il seguente setup (Figura C.1):

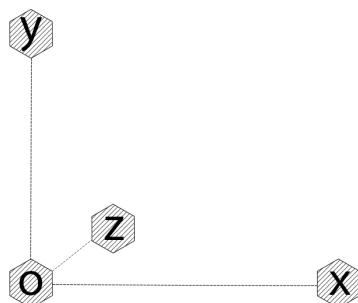


Figura C.1 Assi UWB

Per eseguire l'autocalibrazione basta eseguire il seguente bash script, che si trova nella directory `/home/robot`:

```
>> cd
>> ./autocalibration.sh
```

Una volta terminata la procedura verranno automaticamente impostate le distanze relative tra le varie ancore e si aprirà un'immagine riassuntiva che mostrerà i risultati dell'autocalibrazione. Alle due domande che appariranno sulla command window rispondere entrambe le volte con "n".

## Positioning

Per avviare il nodo che localizza le due tag poste sul robot:

```
>> roslaunch charlie_pkg start_uwb.launch
```

Il tag che deve essere posizionato in testa al robot è quello con "ID = 0x6760". Dato che ad ogni accensione l'ordine con cui vengono letti ed interpretati i tag non è sempre

lo stesso, all'interno dello script è stato implementata la capacità di leggere l'ID delle tag e quindi impostare la tag giusta come testa.

## Acquisizione della mappa

Il robot può essere posizionato in qualsiasi punto dello spazio e con qualsiasi orientazione, non vi è alcuna necessità che gli assi del Lidar (Figura C.2) siano orientati come quelli del setup delle UWB in quanto la posa con cui si inizia ad acquisire la mappa verrà salvata in un file dedicato.

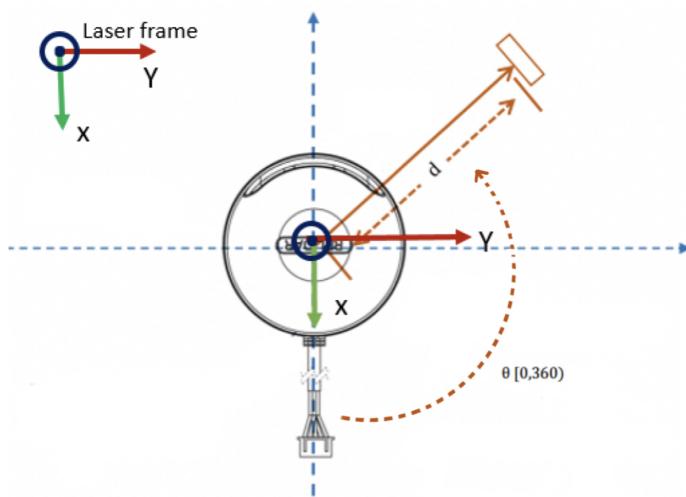


Figura C.2 Assi Lidar

Poiché l'origine della mappa che si andrà ad acquisire corrisponderà al punto in cui viene avviato `hector_slam`, prima di iniziare il processo di mappatura è necessario salvare la distanza relativa tra i sistemi di riferimento di UWB e Lidar e la relativa orientazione. Avviare in primis il sistema UWB tramite il comando:

```
>> rosrun charlie_pkg start_uwb.launch
```

Appena i dati inizieranno a scorrere sullo schermo, avviare il salvataggio della posa corrente con il comando:

```
>> rosrun charlie_pkg save_map_origin.launch
```

Questo avvierà anche la comunicazione seriale, grazie alla quale sarà possibile ricavare l'orientazione del robot rispetto al frame UWB (filtrata dal software a bordo dell'STM<sup>®</sup>).

Una volta che la procedura è completata, chiudere la finestra di lavoro. Si consiglia comunque di segnare la posizione e l'heading del punto in questione poiché, a causa del forte rumore presente sul sistema UWB, potrebbe rivelarsi necessario riposizionare il robot e ripetere tale acquisizione.

A questo punto è possibile acquisire una nuova mappa semplicemente digitando:

```
>> roslaunch charlie_pkg new_map.launch
```

Questo avvierà Lidar, `hector_mapping` e `Rviz` mostrando a schermo lo stato corrente dell'acquisizione della mappa. Una volta soddisfatti del risultato ottenuto, posizionarsi nella cartella in cui si desidera salvare la mappa acquisita (tramite il comando `cd`) e digitare il seguente comando:

```
>> rosrun map_server map_saver -f nomemappa
```

## Localizzazione e navigazione

Nel launch file `localization.launch` inserire il percorso della mappa che si desidera utilizzare e avviare il sistema con i seguenti comandi:

```
>> roslaunch charlie_pkg start_uwb.launch  
>> roslaunch charlie_pkg localization.launch
```

Si aprirà una istanza di `Rviz` sulla quale verrà mostrata la posa stimata del robot e su cui sarà possibile indicare un goal tramite il comando `2DNavgoal`. Per attivare i motori e permettere al robot di spostarsi, pubblicare il seguente messaggio sul topic `start_and_stop`:

```
>> rostopic pub /start_and_stop std_msgs/Float64 "data: 1.0"
```

Per fermare i motori in qualsiasi momento:

```
>> rostopic pub /start_and_stop std_msgs/Float64 "data: 0.0"
```