MASTER DEGREE in EMBEDDED COMPUTING SYSTEMS
A.Y. 2017 - 2018

TurtleBot3 Environment Scanner
Robotics and Human Machine Interfaces

Professori
Antonio Frisoli
Massimiliano Solazzi
Daniele Leonardis

Studenti
Silvio Bacci
Andrea Baldecchi

Indice

1 Introduzione

1.1 Introduzione

L'idea alla base del progetto che abbiamo sviluppato è quella di comandare un Turtlebot 3 in modo automatico o manuale in ambiente virtuale o fisico, utilizzando il sistema di scansione laser "Lidar" di cui è dotato. Il sistema realizzato è in grado di eseguire una scansione dell'ambiente in cui si trova il robot, utilizzando un meccanismo di *collision avoidance* per evitare eventuali ostacoli. Per la realizzazione del progetto abbiamo utilizzato il sistema operativo ROS per gestire il robot, tutto l'insieme di informazioni legate ad esso e all'ambiente di cui facciamo la scansione. Sono stati utilizzati altri due moduli software, Gazebo e Simulink rispettivamente per la simulazione virtuale del sistema e la visualizzazione dei risultati prodotti. Infine è stato effettuato un test utilizzando un Turtlebot 3 fisico.

1.2 ROS

Il sistema operativo ROS è un sistema operativo basato sul paradigma publish/subscribe. L'idea è quella di far cooperare tra di loro diversi nodi che compongono il sistema. Questi nodi pubblicheranno e si sottoscriveranno a topic di loro interesse, gestiti dal ROS master. A questo proposito l'obiettivo del nostro progetto è stato proprio quello di creare un nodo ROS che potesse pubblicare e/o sottoscriversi a diversi topic. Da notare che durante la creazione di questi topic abbiamo definito anche due tipologie di messaggio non predefinite in ROS.

2 Panoramica del progetto

2.1 Scanner node

Scanner node è il nodo ROS che abbiamo implementato nel nostro progetto. Questo nodo permette di accedere ai dati relativi al sensore Lidar, ai dati odometrici e ai comandi inseriti dall'utente utilizzando la tastiera. Inoltre questo nodo elaborerà le informazioni ottenute e le pubblicherà su dei topic a cui il modulo Simulink farà accesso.

2.2 Librerie

Lo scanner node che abbiamo definito utilizza tre librerie da noi implementate. La libreria di controllo, implementata nel file *Controller.cpp,* mette a disposizione un insieme di funzioni utili per realizzare la navigazione del Turtlebot. Oltre ad un insieme di funzioni per la gestione e l'esecuzione di calcoli algebrici espone le funzioni relative all'aggiornamento delle variabili associate ai topic di ROS e una funzione relativa alla collision avoidance. Quest'ultima implementa, in modo semplificato, l'algoritmo descritto in [1], che tratteremo nel dettaglio nei capitoli seguenti.
La seconda libreria definisce tre ROS *publisher* ed espone funzioni utili per la pubblicazione sui vari topic di ROS. Questa è implementata nel file *Publisher.cpp.*
Dualmente, la libreria implementata in *Subscriber.cpp* dichiara quattro subscriber. Espone inoltre le *callback* eseguite al momento della pubblicazione nei relativi topic. Tutte le *callback,* eccetto una, elaborano i dati letti sul relativo topic e li inoltrano su topic diversi in modo tale da mettere a disposizione queste informazioni, trasformate in modo opportuno, all'esterno del sistema operativo. Questo viene fatto per permettere l'interazione tra quest'ultimo e i moduli Simulink e Gazebo che illustreremo nei prossimi capitoli. L a *callback odom* eseguirà inoltre l'algoritmo di collision avoidance implementato nella libreria di controllo prima di effettuare la pubblicazione sul topic associato. La *callback* chiamata *teleop* si limiterà invece a salvare le informazioni lette sul topic in determinate strutture a cui di seguito faranno accesso le funzioni relative alla parte di controllo.

2.3 ROS topics

Andiamo di seguito ad analizzare i topic di cui abbiamo fatto largo uso.

2.3.1 cmd_vel

Questo è un tipo di topic predefinito dal sistema operativo ROS. Contiene le informazioni relative alla velocità del Turtlebot, in particolare conterrà due vettori di tre elementi contenenti velocità lineari ed angolari relative agli assi tridimensionali.

2.3.2 env_constructor

Questo topic, da noi definito, contiene le informazioni relative alla posizione del Turtlebot nello spazio e ai dati acquisiti dal sistema di scansione Lidar. Il formato dei messaggi che vengono scritti in questo topic è definito nel file *tb_cloud_points.msg*. Sono presenti due vettori di 360 elementi *float* ciascuno, che definiscono la posizione di ogni punto acquisito dalla scansione. In aggiunta troviamo due ulteriori variabili *float* che indicano la posizione corrente del robot.

2.3.3 tb_destination

Anche questo topic è stato definito da noi, contiene le informazioni relative alla destinazione che il Turtlebot deve raggiungere, se guidato autonomamente. Il formato dei messaggi ad esso associato viene specificato nell'omonimo file *.msg*. Ogni variabile prevista dal formato è una variabile di tipo *float*:

- *Final_destination_x/y*: due variabili che specificano la posizione del punto di interesse che il robot deve raggiungere.
- *Destination_x/y*: due variabili che specificano la destinazione attuale del robot calcolata dall'algoritmo di collision avoidance.
- *Motion_state:* Variabile che può assumere tre differenti valori indicanti lo stato della fase di contatto calcolata dall'algoritmo di collision avoidance.
- *Tb_x/y:* due variabili che contengono la posizione attuale del TurtleBot.
- *Yaw*: l'orientamento attuale del TurtleBot.
- *Tb_linear_velocity*: Esprime la velocità lineare attuale del robot
- *Tb_angular_velocity:* Analoga alla precedente ma fa riferimento alla velocità angolare.

2.3.4 Topic predefiniti

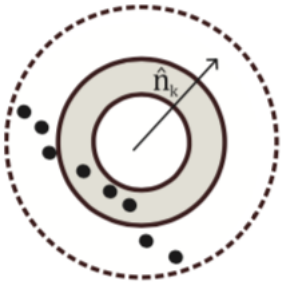
Vengono utilizzati inoltre altri tre topic, predefiniti dal sistema ROS:

- *Scan*: Contiene le informazioni derivate dalla scansione laser.
- *Teleop*: Contiene le informazioni relative alla controllo manuale del Turtlebot da tastiera.
- *Odom*: Contiene le informazioni relative alla odometria al robot utilizzato

3 Collision avoidance

3.1 Introduzione

La fase di collision avoidance viene implementata tramite un opportuno algoritmo descritto presentato in [1]. Questo algoritmo sfrutta il movimento di uno oggetto sferico, chiamato proxy. In particolare a questo saranno associate tre diverse circonferenze, utilizzate per definire la fase di contatto tra il proxy stesso e la superficie individuata dalla nuvola di punti ottenuta utilizzando il sistema di scansione laser Lidar.



Vengono definite tre possibili situazioni:

- Moto libero: Non sono presenti punti tra la circonferenza interna e quella intermedia
- Contatto: E' presente almeno un punto tra la circonferenza interna e quella intermedia
- Trinceramento: E' presente almeno un punto nella circonferenza interna.

L'algoritmo calcolerà quindi l'aggiornamento della direzione di movimento del proxy in base ai seguenti parametri:

- P_k : Posizione del TurtleBot al k-esimo passo
- $\hat{n}_k$: Direzione normale alla superficie individuata dalla nuvola di punti
- u_k : Vettore direzione del proxy

3.2 Algoritmo

Ad ogni spostamento del proxy verrà eseguita un iterazione dell'algoritmo. Analizziamo adesso i vari steps che lo compongono:

1. Si calcola la direzione normale alla superficie individuata dalla nuvola di punti acquisita:

$$\hat{n}'_k = \sum_{i=1}^{N} \frac{P_k - x_i}{\| P_k - x_i \|_2}$$

Dove:
- x_i : Posizione di un punto appartenente alla nuvola
- N : Numero totale di punti della nuvola acquisiti

La direzione $\hat{n}_k$ sarà quindi ottenuto da $\hat{n}'_k$ per normalizzazione.

2. Si valuta la fase di contatto tra il proxy e la nuvola di punti

   1. In caso di moto libero, il movimento del proxy s_k sarà quindi calcolato come:

$$s_k = d_k \frac{u_k}{\| u_k \|_2}$$

   Dove d_k è uno step costante calcolato sperimentalmente. In questo caso quindi il proxy continuerà a muoversi nella direzione associata alla sua direzione originale.

   2. In caso di trinceramento il movimento del proxy sarà quindi espresso come:

$$s_k = d_k n_k$$

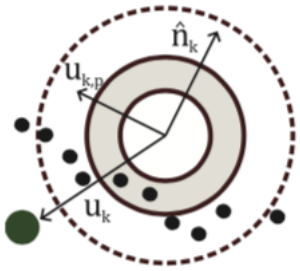
   Il proxy si muoverà quindi nella direzione definita dalla normale calcolata durante lo step 1, allontanandosi dalla superficie. Ovvero effettuando una retromarcia.

   3. In caso di contatto invece avremo la seguente formula:

$$s_k = d_k \frac{u_{k,p}}{\| u_{k,p} \|_2}$$

   Dove $u_{k,p}$ è la proiezione di u_k nel piano definito dalla normale $\hat{n}_k$.

Possiamo apprezzare geometricamente $u_{k,p}$ nella seguente figura:



È importante osservare che la verifica delle fase di contatto avviene considerando le posizioni dei punti acquisiti e $r_{\{1,2,3\}}$, i raggi delle tre circonferenze del proxy.

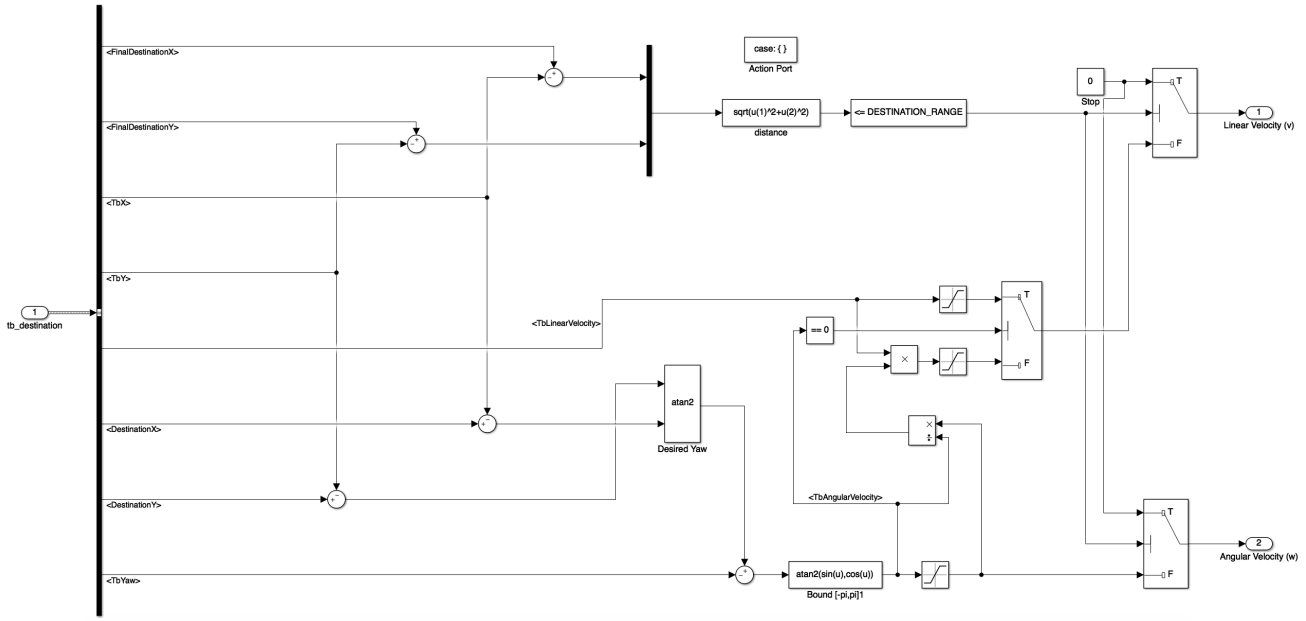
4 Simulink controller

4.1 Introduzione

Per guidare autonomamente il TurtleBot verso un determinato punto di interesse sfruttiamo un controllore proporzionale implementato in Simulink. In funzione della fase di contatto, elaborata dall'algoritmo descritto nel capitolo 3 possiamo distinguere tre comportamenti:

- Moto libero: Il controller si limiterà ad aggiornare lo stato del TurtleBot in modo tale da farlo dirigere verso il punto di interesse.
- Contatto: In questo caso il controller aggiornerà lo stato del TurtleBot in modo tale da evitare il contatto con la superficie individuata dalla nuvola di punti, utilizzando la destinazione temporanea calcolata dall'algoritmo. La velocità lineare del dispositivo viene quindi dimunita per poter evitare l'ostacolo in modo più efficiente e sicuro.
- Trinceramento: Il controller inverte la velocità del TurtleBot realizzando cosi una retromarcia. Il movimento in questa direzione continuerà fino al momento in cui la fase di trinceramento calcolata dall'algoritmo non risulterà essere "moto libero/contatto". A questo punto l'algoritmo di collision avoidance aggiornerà la destinazione in input al controller settandola a quella individuata dalla posizione del punto di interesse.

Da notare che ogni qualvolta il controller calcola la nuova velocità angolare da dare al robot per raggiungere la destinazione, la velocità lineare viene sempre scalata linearmente in modo da poter mantenere la traiettoria desiderata.

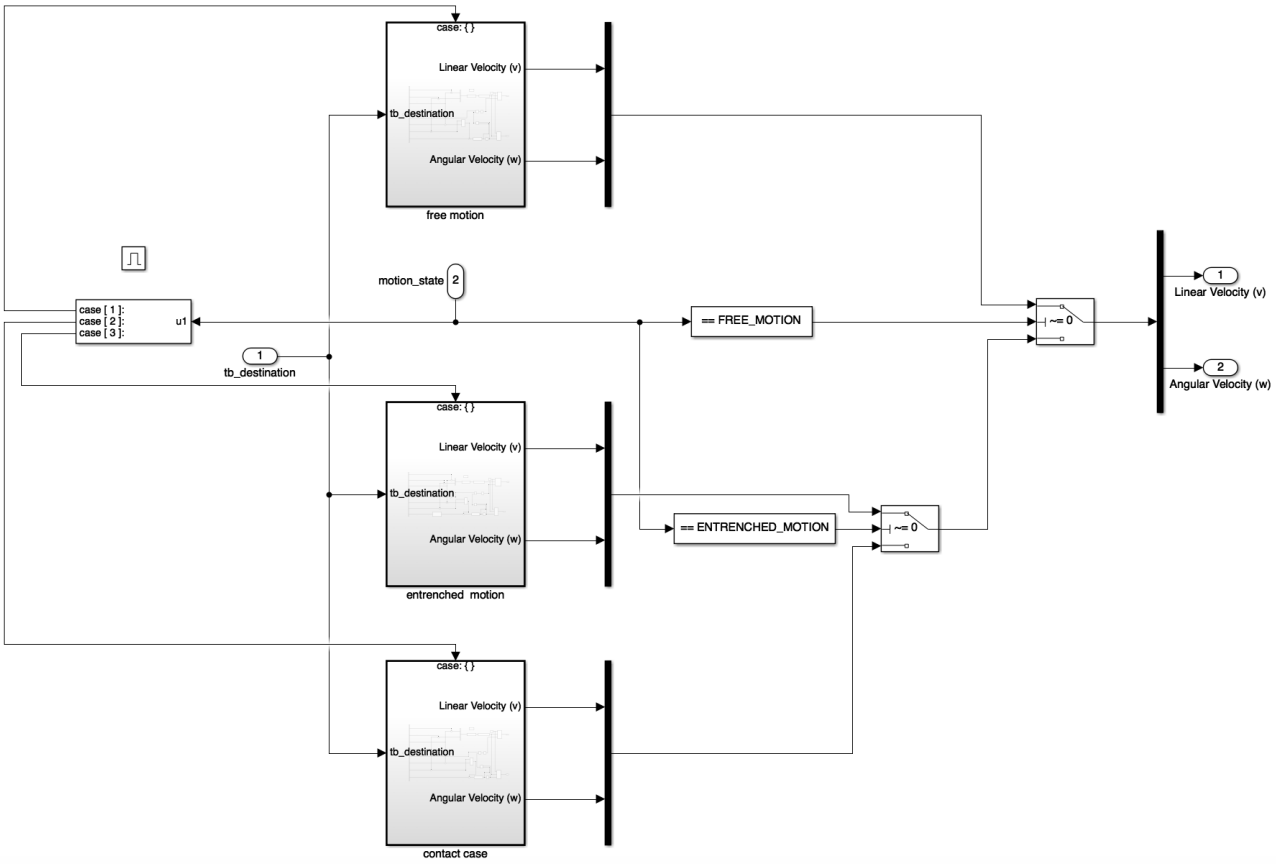


4.2 Gestione della navigazione automatica

Una volta definite le coordinate del punto di interesse possiamo far muovere il TurtleBot. Ogni volta che lo scanner node in ROS accede ai topic */scan* e */odom,* contenenti rispettivamente informazioni relative alla nuvola di punti acquisita e alla posizione/velocità del turtlebot, verrano eseguiti i seguenti passi:

- Esecuzione dell'algoritmo di collision avoidance con la destinazione associata al punto di interesse. In caso di moto libero questo non modificherà la direzione del TurtleBot. In alternativa, la direzione corrente del dispositivo verrà temporaneamente modificata per permettere al dispositivo stesso di evitare un determinato ostacolo
- In funzione della direzione calcolata dall'algoritmo verrà calcolata una nuova destinazione temporanea che verrà pubblicata, insieme alla informazione relativa alla fase di contatto in un apposito topic chiamato */tb_destination*.
- Il controller Simulink, sottoscritto a */tb_destination* acquisirà quindi queste informazioni e in funzioni di queste andrà a modificare in modo opportuno lo stato del TurtleBot effettuando una pubblicazione su un topic denominato */cmd_vel.*

Il modello Simulink generale è il seguente:

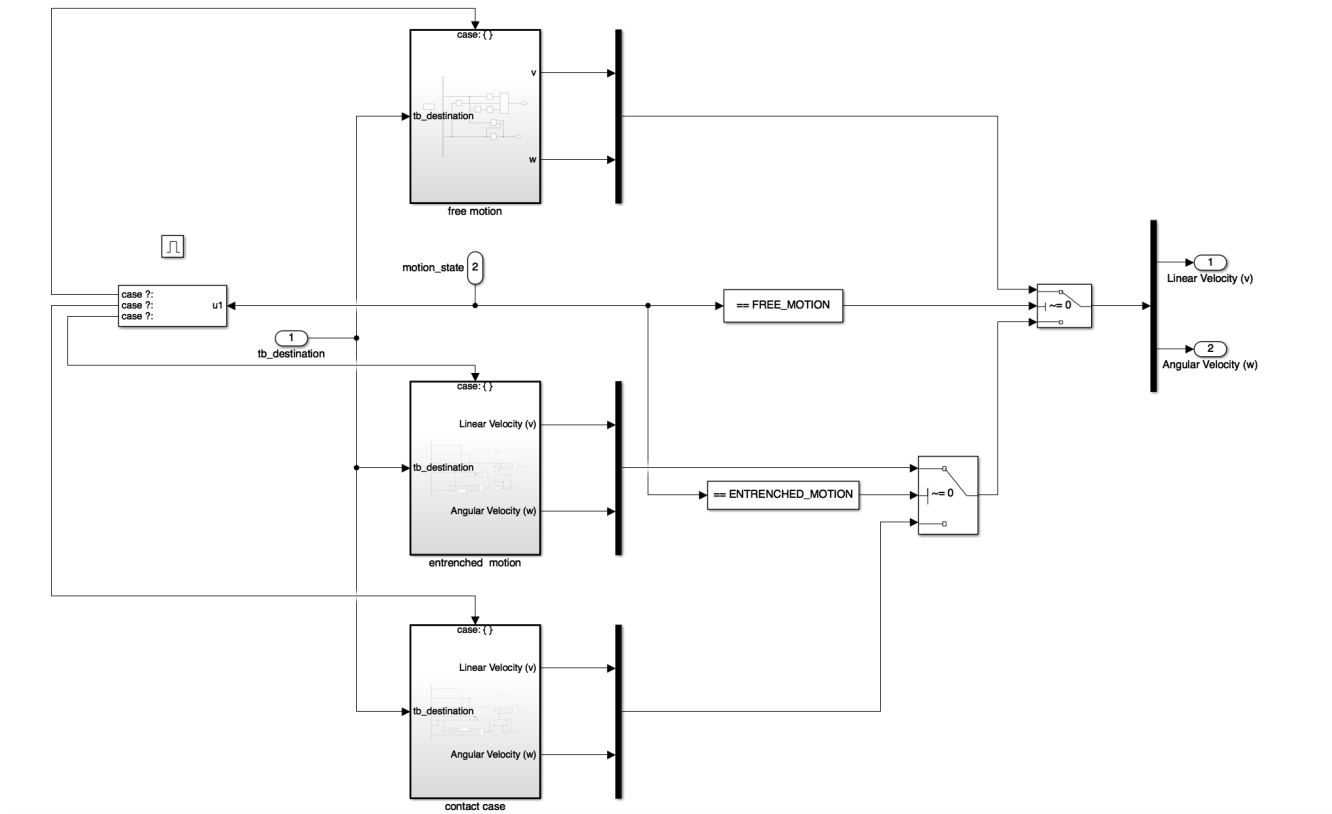


4.3 Gestione della navigazione manuale

Ogni volta che lo scanner node in ROS accede al topic *teleop* andrà a salvare in un apposita struttura la velocità angolare e lineare desiderata dall'utente. Dopodiché le operazione che verranno eseguite sono:

- Esecuzione dell'algoritmo di collision avoidance con la destinazione associata ad un punto calcolato in funzione dello stato del Turtlebot. In particolare considerando la sua posizione, il suo orientamento e la velocità desiderata dall'utente. In caso di moto libero l'algoritmo di collision avoidance non modificherà la direzione del TurtleBot. In alternativa, la direzione corrente del dispositivo verrà temporaneamente modificata per permettere al dispositivo stesso di evitare un determinato ostacolo
- In funzione della direzione calcolata dall'algoritmo verrà calcolata una nuova destinazione temporanea che verrà pubblicata, insieme alla informazione relativa alla fase di contatto in un apposito topic chiamato */tb_destination*.
- Il controller Simulink, sottoscritto a */tb_destination* acquisirà quindi queste informazioni e in funzioni di queste andrà a modificare in modo opportuno lo stato del TurtleBot effettuando una pubblicazione su un topic denominato */cmd_vel*. In particolare, in caso di moto libero il controller si limiterà ad applicare le velocità desiderate dall'utente. Negli altri casi applicherà le velocità utili per poter evitare la collisione.

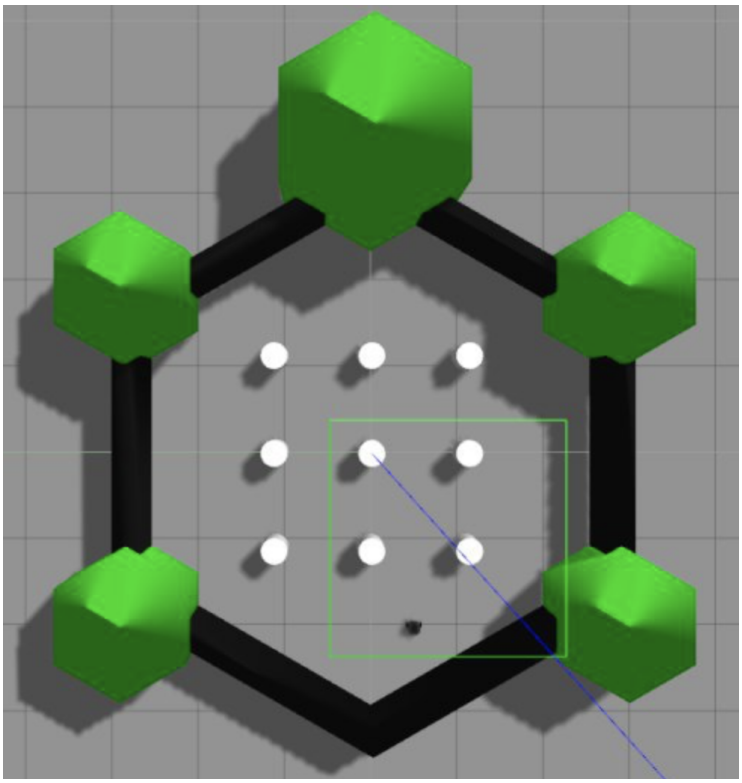
Il modello Simulink generale è il seguente:



5 Environment scanner

5.1 Gazebo

Gazebo è un simulatore che permette di simulare e testare rapidamente algoritmi, robot ed elementi di intelligenza artificiale in scenari fisici con un discreto grado di realismo.
Il modulo software di Gazebo sfrutta quindi l'interfaccia ROS per effettuare una sottoscrizione al topic */cmd_vel*. Grazie a queste, sarà a conoscenza dell'evoluzione nel tempo dello stato del TurtleBot e riuscirà quindi a simularne il comportamento nell'ambiente virtuale. L'ambiente virtuale creato ed utilizzato per la simulazione è sostanzialmente un pentagono nel quale sono stati inseriti una serie di ostacoli di forma cilindrica. Possiamo vederlo nella immagine seguente.



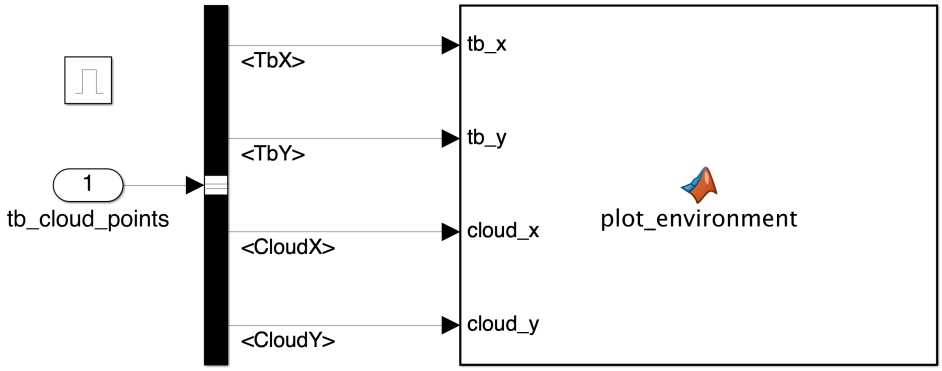
Gazebo si occuperà inoltre di simulare il funzionamento del sistema di scansione laser Lidar montato sulla struttura del TurtleBot. I dati prodotti da quest'attività di simulazione saranno quindi pubblicati sul topic */scan* in modo tale da renderli disponibili per le fasi di gestione della navigazione trattata nei capitoli precedenti e per l'attività di costruzione dell'ambiente in Simulink.

5.2 Environment constructor

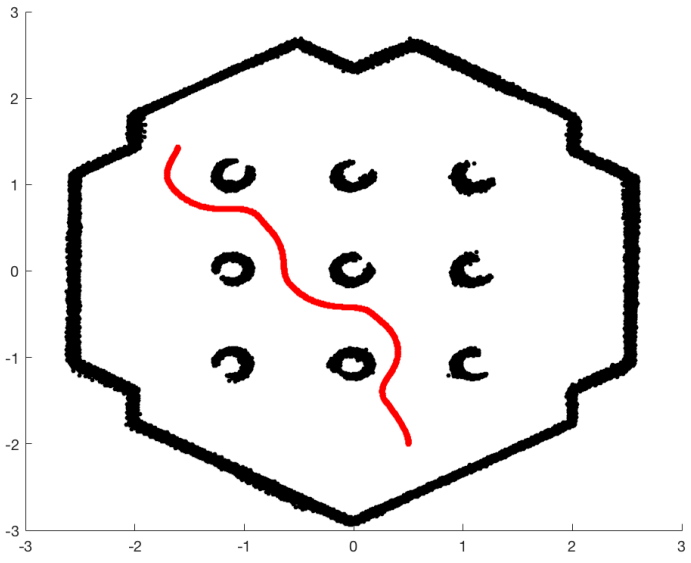
Abbiamo creato un apposito modello Simulink per riuscire a disegnare, utilizzando lo scatter plot di MATLAB, l'ambiente virtuale definito usando il simulatore Gazebo. Questo modello Simulink utilizzerà quindi l'interfaccia ROS di MATLAB per effettuare una sottoscrizione al topic */env_constructor*. Questo conterrà in ogni istante di tempo:

- Dati prodotti dal sistema Lidar, ovvero un insieme di 360 distanze, ognuna associata ad un determinato angolo di rotazione.
- La posizione del Turtlebot

Il modello Simulink è il seguente:



Un esempio del risultato prodotto è il seguente:



Referenze

[1] : A Proxy Method for Real-Time 3-DOF Haptic Rendering of Streaming Point Cloud Data Fredrik Ryde´ n, Student Member, IEEE, and Howard Jay Chizeck, Fellow, IEEE.