Progetto

Intelligent Systems and Robotics Laboratory

Indice generale

1 Obiettivo del progetto

L'obiettivo del progetto consiste nel far esplorare al robot a quattro ruote FreeNove 4WD un labirinto la cui struttura non è nota al robot, in modo autonomo e intelligente. L'algoritmo che abbiamo ideato ha qualche caratteristica in comune con l'algoritmo di ricerca in profondità DFS. Il robot inizia la sua esplorazione da un punto iniziale, chiamato radice o stato iniziale e, in base ai valori dei sensori di cui è dotato, l'algoritmo elabora i dati e decide la direzione e verso in cui si deve muovere il robot. Se il robot trova un vicolo cieco dal quale non è più possibile proseguire in avanti, l'algoritmo deciderà di far ripercorrere al robot il percorso intrapreso nel verso opposto fino a quando non trova un nuovo percorso da esplorare. L'esplorazione si conclude quando il robot trova il punto finale del labirinto.

Nella presente documentazione, Agente, Robot e macchina assumono lo stesso significato.

2 Software e linguaggi utilizzati

Il progetto è stato realizzato utilizzando i seguenti software/servizi e linguaggi di programmazione.

Software/servizi:

- IDE: Pycharm e Spyder
- Simulatore di robotica 3D: CoppeliaSim
- Sistema di controllo di versione: GitHub

Linguaggio di programmazione:

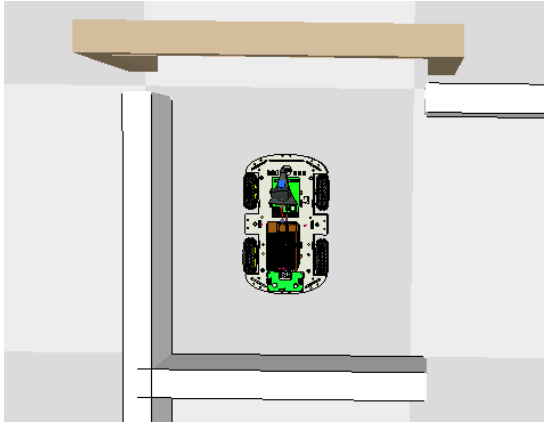
- Python

3 Vincoli di progetto

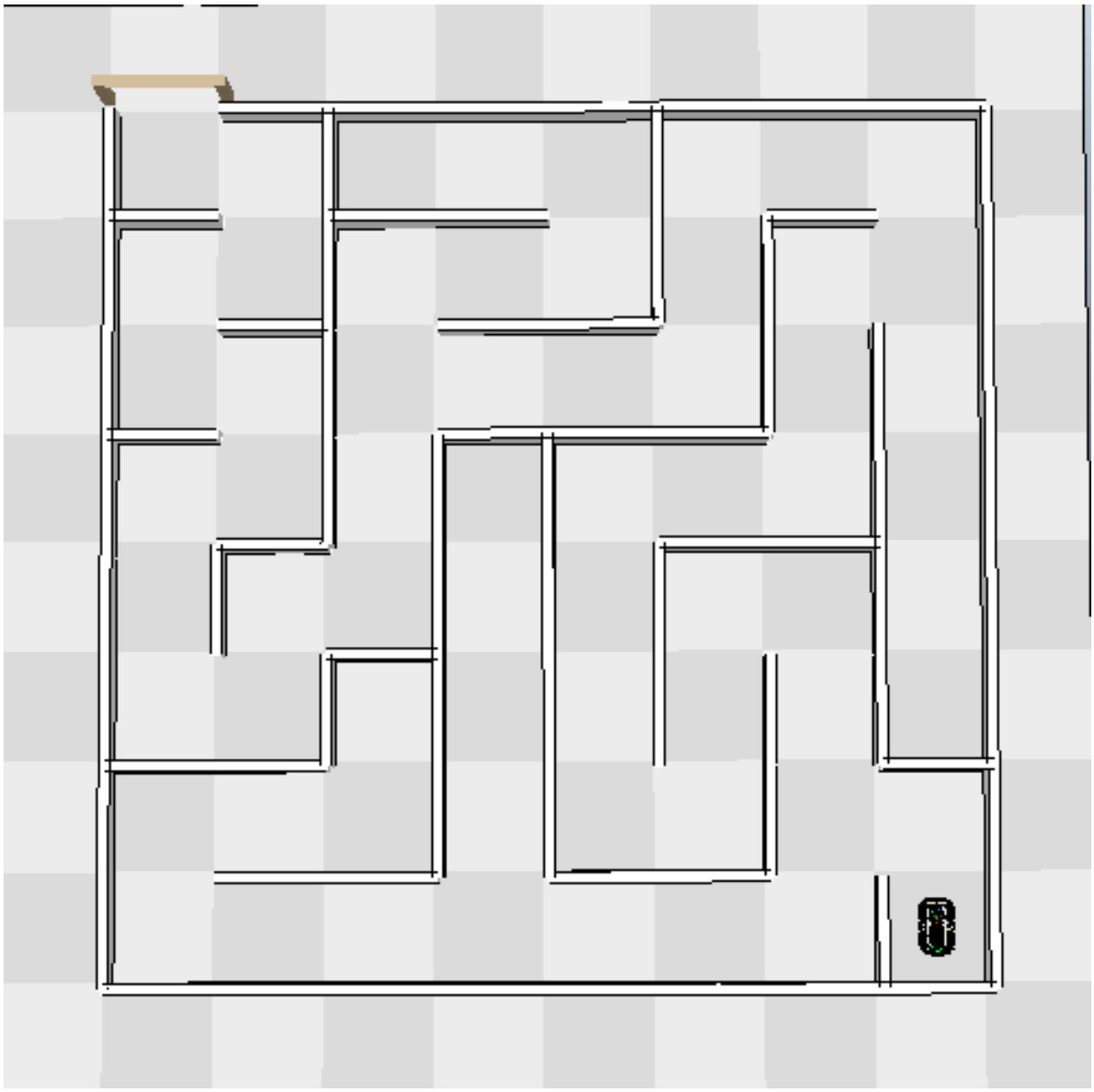
3.1 Labirinto

Il labirinto deve avere una struttura tridimensionale che deve rispettare i seguenti vincoli:

1. Struttura statica e non può mutare nel tempo.
2. Un solo punto iniziale (radice/root).
3. Un solo punto finale (uscita) dal labirinto (final).
4. Può essere rappresentato attraverso un albero semplice:
   a) Non ci sono cicli.
   b) Non è possibile passare per uno stesso punto se si esplora in profondità.
5. Presenza di vicoli ciechi.
6. Percorso formato da muri che formano angoli di $90°$ e $180°$.
7. Esiste un percorso più breve che permette all'agente di andare dallo stato iniziale allo stato finale.
8. Labirinto sconosciuto all'agente e quindi non può sapere a priori il percorso più breve dall'ingresso fino all'uscita (detto anche punto finale).
9. Uscita sconosciuta all'agente.
10. L'agente tiene traccia delle decisioni che ha eseguito e lo fa senza la nozione di tempo.
11. Al fine di prendere una decisione informata riguardo a quale azione eseguire, il robot deve essere in grado di osservare ciò che è attorno a lui.
12. Nella robotica virtuale:
    12.1. In prossimità dell'uscita del labirinto, è presente un "gate" costituito da sensori di prossimità che consentono di rilevare la presenza o meno del robot. Se il sensore rileva il robot vuol dire che quest'ultimo è riuscito a trovare l'uscita e l'algoritmo di esplorazione termina.
    12.2. Distanza tra due muri deve essere almeno di 0.45 m.
    12.3. Spessore deve essere almeno di 0.10 m.
13. Fisico:
    a) Distanza tra due muri deve essere almeno di ...
    b) Spessore deve essere almeno di ....



Gate



Maze_1

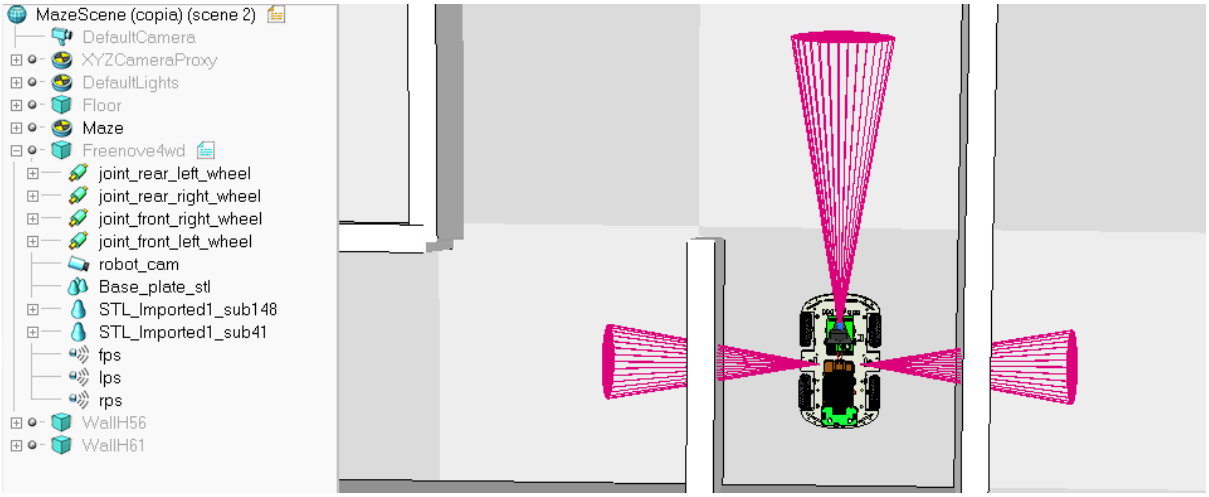
3.2 Robot

Il robot è il modello in 3D del FreeNove 4WD.

Esso è costituito dai seguenti sensori di prossimità:

- Front: posizionato nella parte frontale del Robot
- Left: posizionato nella parte di sinistra del robot e forma un angolo di 90° con il sensore Front
- Right: posizionato nella parte di destra del robot e forma un angolo di 90° con il sensore Front

Scrivere distanze, tipo di sensori, distanza minima di sicurezza che il robot deve rispettare.



4 Algoritmo

Il Controller contiene l'algoritmo principale per prendere decisioni e dare comandi al robot. Tutto ciò avviene solo dopo che sono stati analizzati ed elaborati i valori dei sensori di prossimità, lo stato (State), la posizione della macchina (Position), la modalità di esplorazione (Mode) e l'albero (Tree) del labirinto. Le decisioni si traducono in comandi che devono essere eseguiti dal physical_body. L'algoritmo si basa sull'architettura SENSE, THINK, ACT che verrà analizzata in modo più dettagliato successivamente.

Adesso andremo a spiegare quali valori possono assumere State, Position, Mode capendo l'importanza che hanno per il corretto funzionamento dell'algoritmo.

State, Position, Mode sono classi di tipo Enum.

1) Lo stato *State* del robot può assumere diversi valori come:

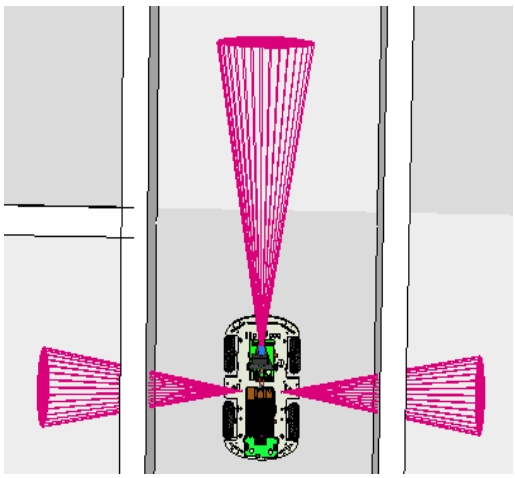
- STARTING = -1
- STOPPED = 0
- RUNNING = 1
- ROTATING = 2
- SENSING = 3

STARTING: Primo stato che assume il robot, nella fase di inizializzazione. Non può tornare più in questo stato durante l'esplorazione.

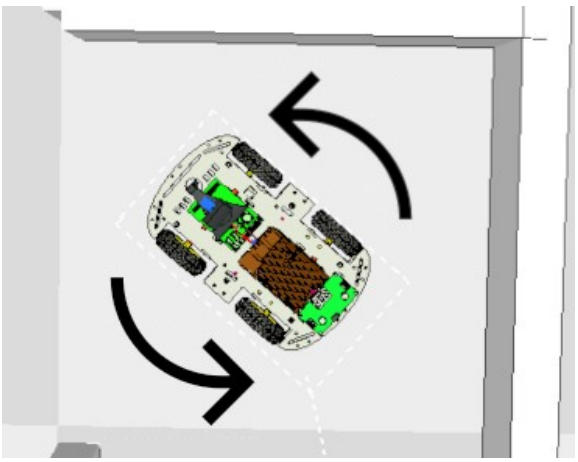
STOPPED: Il robot ha i motori spenti ed è fermo.

RUNNING: Robot in corsa durante il quale analizza continuamente, tramite i sensori, lo spazio circostante.

ROTATING: Stato in cui il Robot sta ruotando su sé stesso (asse z).



RUNNING



ROTATING

SENSING: Stato in cui il Robot è fermo e analizza, tramite i sensori, lo spazio circostante. Stato cruciale in cui avvengono importanti operazioni tra le quali è presente l'aggiornamento dell'albero del labirinto.

2) La posizione *Position* della macchina non è la posizione data dalle coordinate, ma assume un altro significato. Può avere valori come:

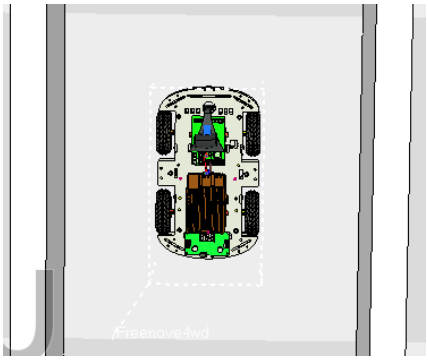
   - UNKNWON $= 0$
   - INITIAL $= 1$
   - CORRIDOR $= 1$
   - JUNCTION $= 2$

UNKNWON: Posizione sconosciuta al robot. Il robot deve analizzare lo spazio circostante tramite i sensori in modo da potersi posizionare nella posizione corretta. Dato che i sensori del robot sono Front, Left e Right è necessario ruotare il robot per capire se dietro di esso c'è un muro o meno. Una volta fatte le dovute rotazioni, la parte posteriore deve essere contro il muro e il valore di Position diventa INITIAL.

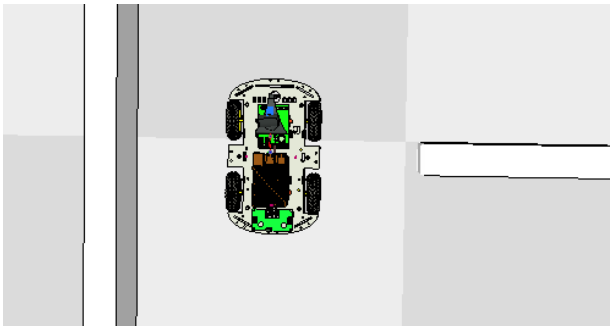
INITIAL: Posizione iniziale, fase di inizializzazione. Il robot deve assumere una posizione che deve rispettare i seguenti vincoli:

- La parte posteriore del robot deve essere contro il muro.
- Il robot non può essere posizionato in una giunzione/intersezione con 4 direzioni/strade disponibili.
- Il robot non può essere posizionato in mezzo al corridoio dove è possibile andare in due sensi (avanti e indietro)

CORRIDOR: Il Robot si trova nel corridoio. Definizione: *spazio ristretto dove la macchina può andare avanti o indietro ma non può svoltare né a sinistra né a destra*.



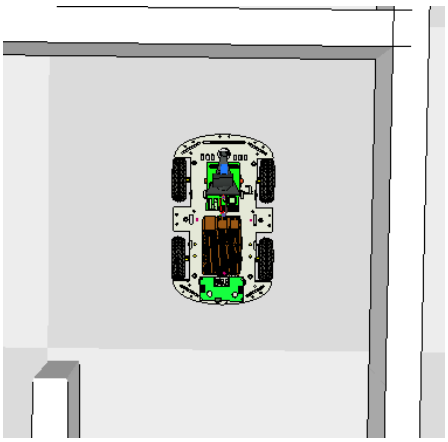
CORRIDOR



CORRIDOR

Robot in un corridoio. Bordo muro di destra considerato come muro di un corridoio.

JUNCTION: Il Robot si trova in una giunzione. Definizione: *spazio dove la macchina può andare a destra e a sinistra oltre alla possibilità di andare avanti e indietro. La condizione necessaria e sufficiente è che può andare a sinistra o a destra.*



JUNCTION

3) La modalità *Mode* di esplorazione:

- EXPLORING $= 0$
- ESCAPING $= 1$

EXPLORING: Modalità di default. La macchina si trova in questa modalità quando deve esplorare spazi non conosciuti del labirinto. Durante questa fase, l'algoritmo crea in modo progressivo dei nodi che appenderà all'albero in modo sistematico. Tale albero mappa il labirinto in base a determinate condizioni che saranno spiegate dettagliatamente più avanti.

ESCAPING: Modalità che viene attivata quando la macchina trova un vicolo cieco. L'algoritmo deciderà di far ripercorrere al robot il percorso fatto fino ad ora fino a quando non trova un nuovo percorso da esplorare e, in tal caso, verrà riattivata la modalità EXPLORING.

Una volta che il Controller ha analizzato lo stato effettivo in cui si trova il robot, lo spazio circostante ed elaborato dati, prende decisioni su quali movimenti la macchina deve svolgere. Il Controller quindi invia al Physical_body i comandi e azioni che poi si traducono in movimenti fisici della macchina. Una nota

importante è che ogni comando Command è affiancato da un'azione di tipo Compass e il solo comando non è sufficiente a far svolgere un movimento al robot. Infatti il Controller invia al physical_body la seguente coppia:

```
[command, action]
```

Es:

```
[<Command.ROTATE: 2>, <Compass.OVEST: 180.0>]
```

Abbiamo realizzato la classe Compass per la definizione dei punti cardinali e per alcune funzioni utili per la conversione di stringhe con il tipo Compass e viceversa.

Compass:

- NORD = 90.0
- SUD = -90.0
- EST = 0.0
- OVEST = 180.0

La classe Command è costituita dai seguenti comandi:

- START = -1
- STOP = 0
- RUN = 1
- ROTATE = 2
- GO_TO_JUNCTION = 3
- BALANCE = 4

START: Primo comando inviato all'accensione della macchina e quindi alla prima esecuzione dello script in python.

STOP: Utilizzato per fermare la corsa della macchina. Utilizzato nei seguenti casi:

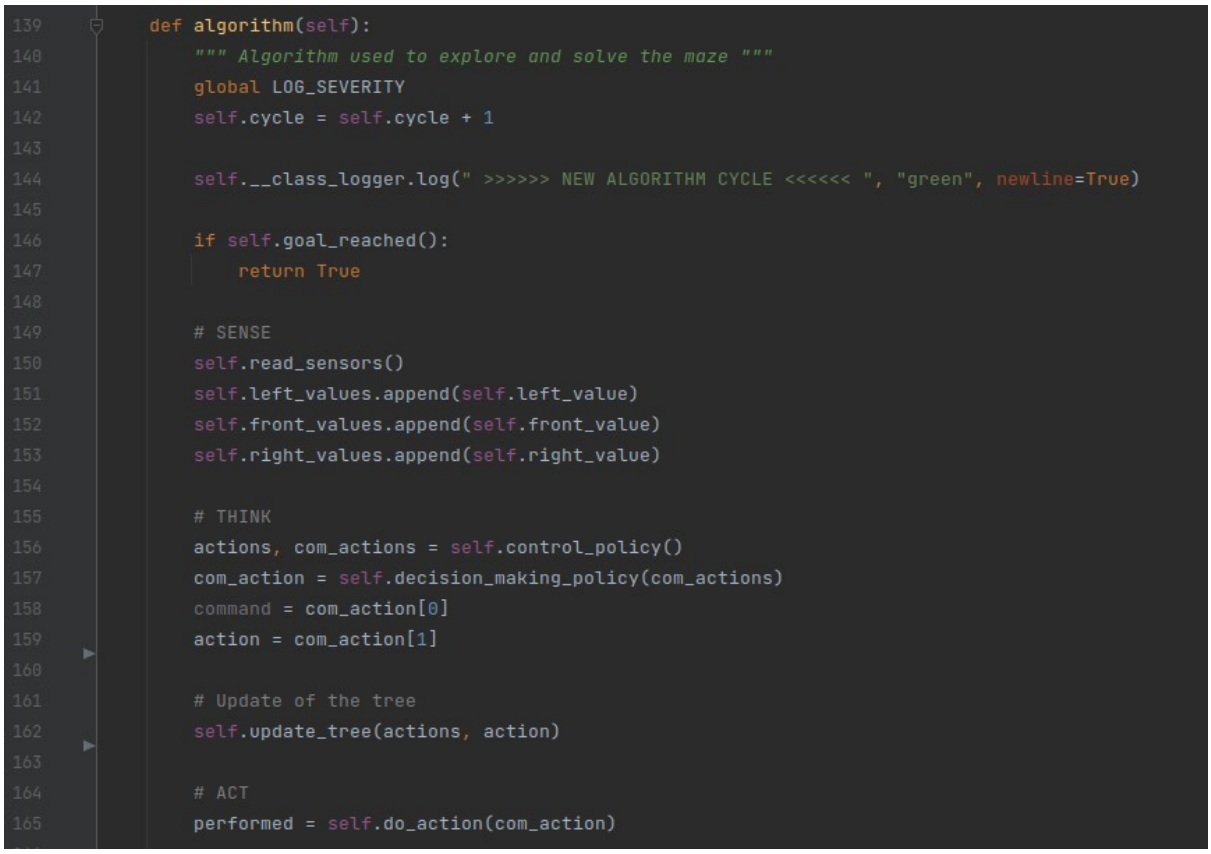
- Vicinanza di un ostacolo nella direzione frontale del robot
- Robot posizionato in una giunzione

RUN: Comando per far muovere il robot in avanti

ROTATE: Usato per far ruotare il robot su sé stesso

GO_TO_JUNCTION: Usato per far posizionare il robot al centro di una giunzione

BALANCE: Quando una parte laterale del robot si trova troppo vicino ad un muro, il Controller invia questo comando per far posizionare il robot ad una distanza minima di sicurezza dal muro per evitare scontri non voluti. È possibile attivare e disattivare la modalità di bilanciamento semplicemente modificando il file *config.conf*



```
139     def algorithm(self):
140         """ Algorithm used to explore and solve the maze """
141         global LOG_SEVERITY
142         self.cycle = self.cycle + 1
143 
144         self.__class_logger.log(" >>>>>> NEW ALGORITHM CYCLE <<<<<< ", "green", newline=True)
145 
146         if self.goal_reached():
147             return True
148 
149         # SENSE
150         self.read_sensors()
151         self.left_values.append(self.left_value)
152         self.front_values.append(self.front_value)
153         self.right_values.append(self.right_value)
154 
155         # THINK
156         actions, com_actions = self.control_policy()
157         com_action = self.decision_making_policy(com_actions)
158         command = com_action[0]
159         action = com_action[1]
160 
161         # Update of the tree
162         self.update_tree(actions, action)
163 
164         # ACT
165         performed = self.do_action(com_action)
```

Nel SENSE avviene la lettura dei valori dei sensori di prossimità: *Front*, *Left*, *Right*.

Nel THINK ci sono principalmente due funzioni:

- **`control_policy()`**: in base ai valori dei sensori, lo stato effettivo del robot e l'albero del labirinto restituisce due liste:
  - **`actions`**: lista di azioni di tipo **`Compass`** (ovvero bussola) ed è usata solo per aggiornare l'albero del labirinto.
  - **`com_actions`**: abbreviazione di command_actions. È una lista di liste di due elementi: uno di tipo **`Command`** e l'altro di tipo **`Compass`**. In altre parole, contiene una lista di coppie *`[command, action]`* e solo una di queste viene realmente eseguita dal robot.

- **`decision_making_policy(com_actions)`**: data la lista **`com_actions`** precedentemente descritta, decide la coppia *`[command, action]`* che deve essere eseguita dal robot. La scelta viene effettuata considerando la lista di priorità **`priority_list`** che contiene in ordine di priorità i punti cardinali Nord, Sud, Est, Ovest. L'ordine della lista può essere modificato aggiornando il file *config.conf.*

Diagramma degli stati:

……

……

lo stato effettivo e reale del robot è ottenuto dalla combinazione di:

- State
- Position
- Mode