TESINA DI ROBOTICA

RescueRobot and LostRobots Scenario

Prof. Antonio Chella
Prof.ssa Valeria Seidita
Ing. Arianna Pipitone

Alessandro Di Giovanni

Ingegneria Informatica
Anno Accademico 2021/22

Sommario

1. Introduzione

Il seguente progetto si propone di realizzare uno scenario in cui, idealmente, un robot disperso in grado solamente di spostarsi da un punto a un altro, richiede e riceve assistenza da un altro robot che gli indica il percorso verso il punto di destinazione.

Più nello specifico viene presentato un ambiente multi agente inizialmente sconosciuto in cui sono presenti quattro robot: un robot principale denominato “RescueRobot”, e tre robot secondari, individuati univocamente dal loro colore, denominati “LostRobot”.

Inizialmente, “RescueRobot”, dotato di tutti gli strumenti fisici e algoritmici necessari, esplora l’ambiente sconosciuto, ne genera una mappa e si mette in ascolto per eventuali richieste di assistenza da parte di altri robot.

In modo da rendere il tutto più automatizzato e alleggerire il peso computazionale sul robot principale, dopo l’esplorazione vengono generati in punti casuali della mappa i robot dispersi.

I “LostRobot”, considerati robot “ciechi” quindi di fatto in grado solo di muoversi e comunicare, dopo aver inviato una segnalazione di aiuto ricevono da “RescueRobot” un percorso specifico per andare dalla loro posizione iniziale a quella finale.

Nella realizzazione del progetto sono state applicate quante più possibili nozioni teoriche e pratiche acquisite durante il corso di robotica.

2. Oggetti della simulazione

Il progetto è stato interamente sviluppato tramite l'utilizzo del simulatore robotico *Webots* ***R2021b***. Il simulatore mette a disposizione un ambiente di sviluppo dove è possibile creare ambienti complessi in cui introdurre diversi tipi di robot realmente esistenti, per poi infine trasferire eventualmente su una macchina reale il risultato dello studio condotto sul simulatore.

Nel seguito verranno illustrati i "protagonisti" della simulazione riguardante il progetto.

2.1 Robot

All'interno della simulazione troviamo sicuramente i quattro robot: *RescueRobot* e i tre *LostRobot*. Il modello base di robot scelto sia per *RescueRobot* che per *LostRobot* è *TIAGo Base* progettato dalla *PAL Robotics*, opportunamente modificato per adattarsi agli scopi dei rispettivi robot.

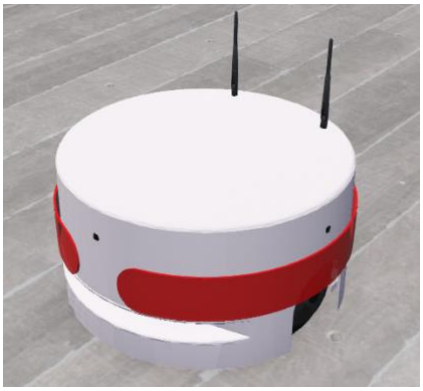
2.1.1 Caratteristiche generali

Di seguito le caratteristiche del *TIAGo Base:*

- Diametro: 0.544 m
- Altezza: 0.300 m
- Velocità massima: 1.5 m/s
- Ruote motorizzate: 2
- Diametro ruota: 0.0986 m

2.1.2 RescueRobot

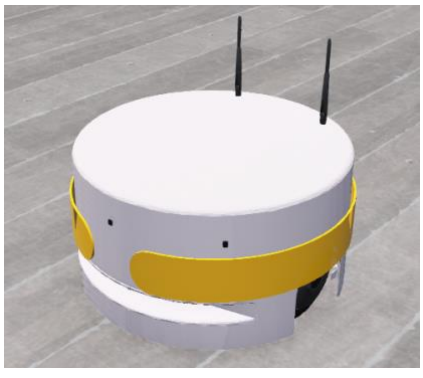
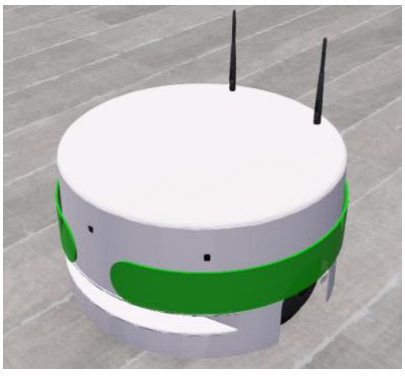
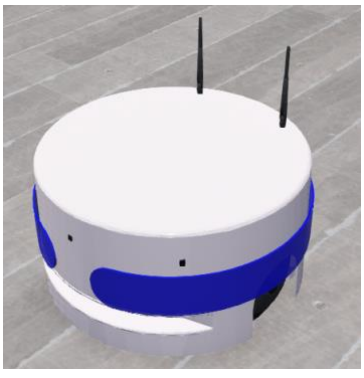
Rispetto al modello base sono stati aggiunti nuovi sensori e dei moduli per la comunicazione tra robot. In particolare *RescueRobot* presenta:



- Due sensori di posizione di tipo rotazionale, utilizzati nei calcoli odometrici;
- Quattro sensori di distanza di tipo laser, uno per ogni direzione cardinale, quindi in grado di rilevare oggetti di fronte, dietro, a destra e sinistra posizionati a una distanza massima di 1.5m;
- Una bussola, utilizzata per ottenere l'orientazione del robot;
- Tre coppie emettitore/ricevitore, per consentire la comunicazione con gli altri robot tramite segnali radio in maniera parallela;

2.1.3 LostRobot

Il modello è progettato per scopi diversi rispetto a *RescueRobot* e di conseguenza cambiano gli strumenti forniti. Più nello specifico in *LostRobot* troviamo:

- Due sensori di posizione di tipo rotazionale, utilizzati nei calcoli odometrici;

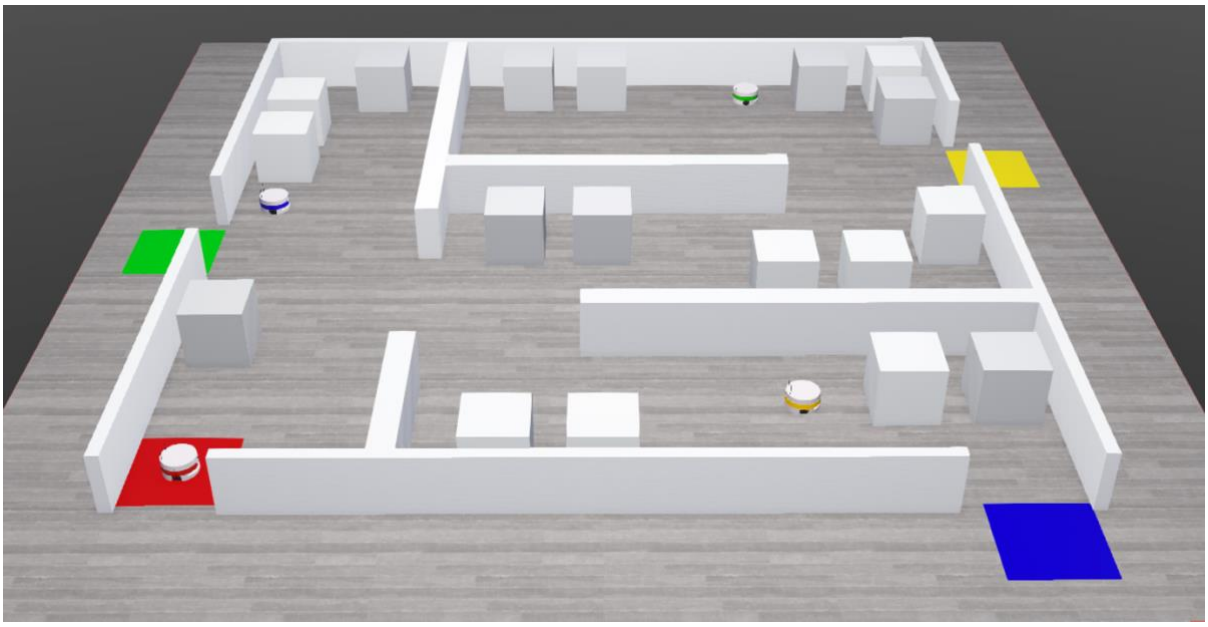
- Due sensori di distanza di tipo laser, uno per rilevare oggetti frontali e uno per oggetti sulla diagonale sinistra, entrambi utilizzati nella *collision avoidance* dei *LostRobot* come verrà spiegato nel seguito;
- Una bussola, utilizzata per ottenere l'orientazione del robot;
- Un sensore GPS per stabilire la posizione iniziale;
- Una coppia emettitore/ricevitore per comunicare con *RescueRobot* tramite segnali radio;

2.2 Ambiente

Come verrà ampiamente descritto in seguito, l'ambiente è progettato come una matrice in cui ogni cella può essere percorribile, rappresentare un ostacolo o un robot disperso. Viene quindi proposto un possibile ambiente con una configurazione di ostacoli, e di punti di arrivo dei singoli *LostRobot* dove testare che tutto funzioni correttamente.

Tuttavia, non c'è nessun vincolo di: configurazione degli ostacoli, tipo di ostacoli, scelta dei punti di partenza e arrivo dei singoli *LostRobot*, punto di partenza di esplorazione del *RescueRobot*.

L'unica limitazione è che devono essere rispettati i punti delle celle della matrice che rappresenta l'ambiente.



Nella configurazione di base si è scelto di utilizzare semplici ostacoli per non appesantire maggiormente la CPU della macchina su cui eseguire la simulazione, già impegnata nell'eseguire quattro controllori.

Le celle colorate indicano il punto di destinazione del LostRobot dello stesso colore.

3. Software

Per la tipologia del progetto un linguaggio di programmazione con paradigma a oggetti è stato valutato come il più efficace e considerando il livello di conoscenza dei vari linguaggi, si è scelto di sviluppare il progetto in Java, con ovviamente l'ausilio delle librerie proprie di Webots.

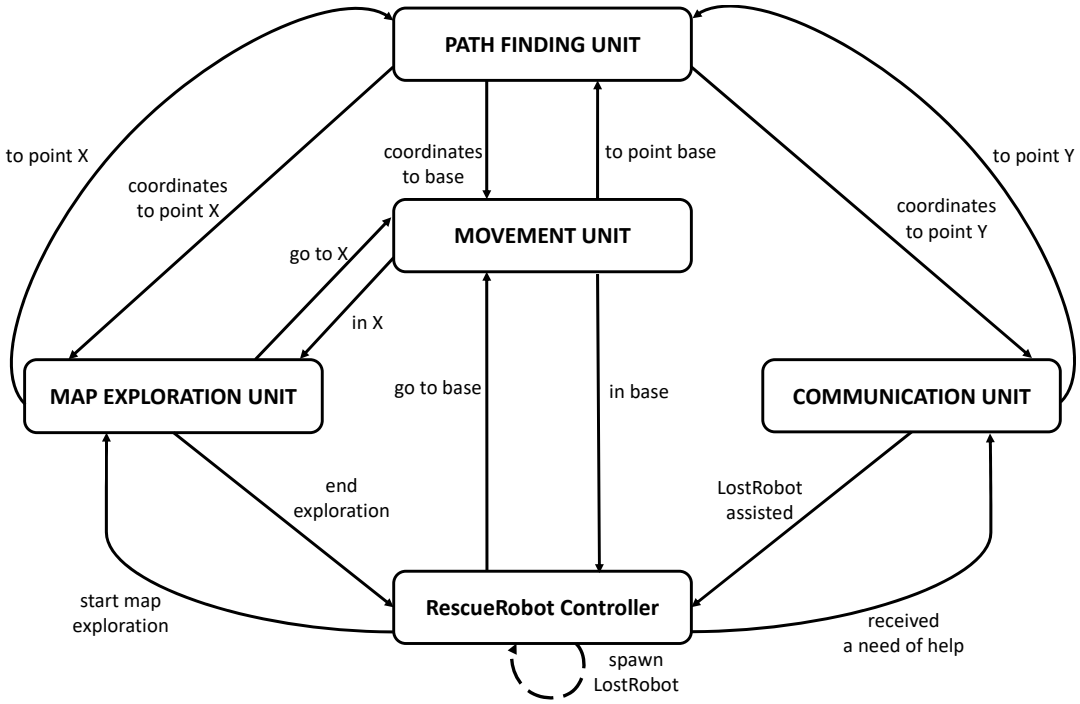
I controllori principali sono due: uno per RescueRobot e uno per LostRobot, l'ultimo è unico per tutti i LostRobot e si adatta allo specifico robot disperso in fase di esecuzione.

3.1 RescueRobot Controller

Il controllore di *RescueRobot* si occupa di gestire quattro unità fondamentali funzionali ai suoi obbiettivi e in più di generare in punti casuali della mappa i *LostRobot*:

1. ***MovementUnit***, responsabile del movimento del robot e del suo posizionamento all'interno della mappa;
2. ***MapExplorationUnit***, responsabile dell'esplorazione dell'ambiente sconosciuto e della generazione e aggiornamento della mappa dello stesso;
3. ***PathFindingUnit***, responsabile della ricerca di un percorso da un punto iniziale ad un punto finale all'interno della mappa;
4. ***CommunicationUnit***, responsabile del corretto scambio di messaggi con i *LostRobot*;

Il comportamento generale è riassunto dal seguente automa a stati finiti (FSA):



3.1.1 MovementUnit

L'unita "MovementUnit", come già accennato, si occupa della cinematica del robot e contestualmente di aggiornare la sua posizione all'interno della mappa.

Questa unità implementa il modello di movimento basato su odometria, cioè che sfrutta le informazioni provenienti da specifici sensori del robot, in particolare dai *PositionSensor* presenti sulle ruote e dalla bussola interna, per poter calcolare e gestire il movimento.

Dato il robot nello **stato iniziale**:

$$(x, y, \theta)$$

dove *x* e *y* sono le coordinate cartesiane rispetto al punto di riferimento assoluto e θ è la sua orientazione attuale rispetto al nord;

Per andare allo **stato finale**:

$$(x', y')$$

Il robot deve eseguire una **rotazione** pari a:

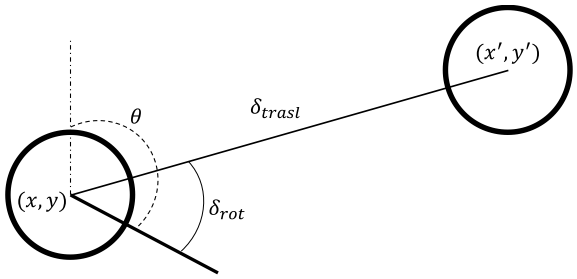
$$\delta_{rot} = (atan2(y' - y, x' - x) - \theta) + \frac{\pi}{2}$$

La funzione *atan2(y, x)* restituisce l'angolo, in coordinate polari, di un punto *(x, y)*; in questo caso calcola l'angolo tra la diagonale che collega i due stati, iniziale e finale, e l'asse dell'ascisse idealmente coincidente con zero (gradi o radianti). Per ottenere la rotazione corretta, a questo valore bisogna aggiungere l'angolo θ di partenza, e uno sfasamento di $\pi/2$ dato che per *RescueRobot* lo zero coincide con il nord cardinale, quindi con l'asse delle ordinate, e non con l'asse delle ascisse.

Infine il robot esegue una **traslazione** pari a:

$$\delta_{trasl} = \sqrt{(x' - x)^2 + (y' - y)^2}$$

che rappresenta la distanza euclidea tra i due punti.



Dato che il robot si muove in un ambiente simulato non è stato introdotto alcun tipo di rumore, i piccoli errori di posizione dovuti alla natura discreta delle informazioni sensoriali vengono gestiti e corretti dalla *MapExplorationUnit* come verrà illustrato in seguito.

3.1.2 MapExplorationUnit

La *"MapExplorationUnit"* è l'unità principale dato che implementa un algoritmo di SLAM che è fondamentale per le funzionalità e gli scopi di *RescueRobot*.

Innanzitutto, l'ambiente viene discretizzato in una matrice 9x9 dove ogni cella, di dimensione 2.25m^2, può assumere quattro differenti valori che ne indicano lo stato:

- **F**, indica una cella libera, quindi percorribile da un robot;
- **O**, indica una cella che presenta un ostacolo di qualunque genere, quindi non percorribile da un robot;
- **S**, indica una cella di stato sconosciuto, cioè una cella non ancora esplorata o inaccessibile e della quale quindi non interessa lo stato;
- **R**, indica nella mappa la cella in cui era presente un LostRobot.

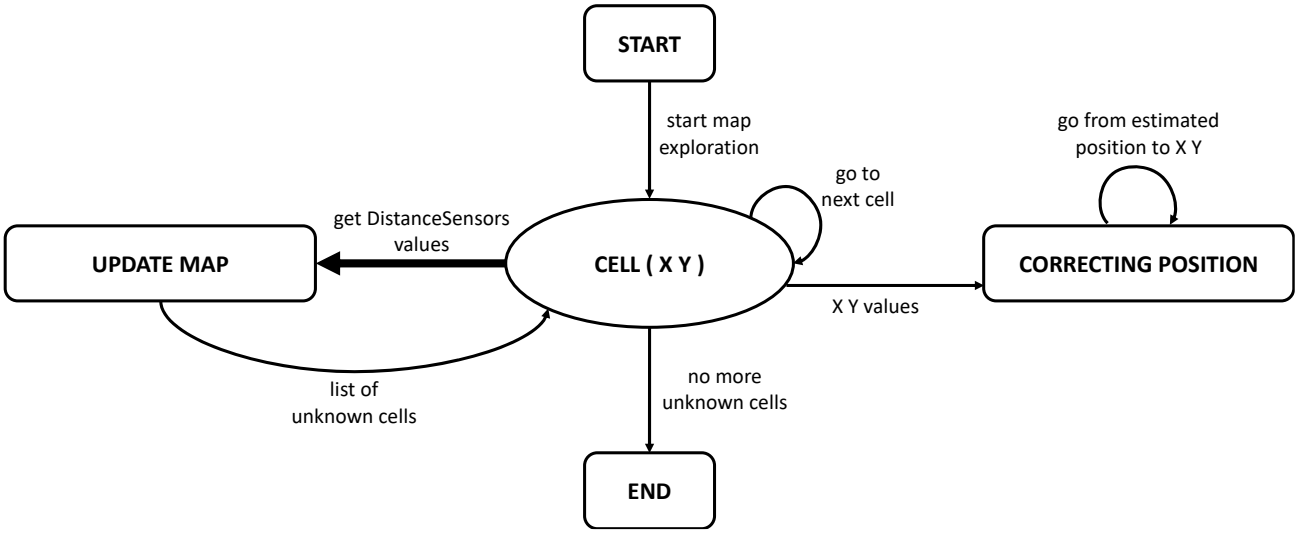
In aggiunta agli stati, durante l'esplorazione si tiene anche traccia delle celle visitate, ovvero in cui il robot è transitato.

Il problema di modellare una mappa dell'ambiente e contemporaneamente stimare la posizione del robot all'interno viene definito come **SLAM** (Simultaneous Location and Mapping), che di fatto consiste nel trovare la seguente funzione di densità di probabilità:

$$p(x_t, m \mid z_{1:t}, u_{1:t})$$

ovvero stimare la posizione e la mappa date una serie di misure sensoriali e un insieme di movimenti.

Tuttavia, in questa simulazione, si è ridotto al minimo il numero di errori e disturbi sia sui sensori sia sugli attuatori, di conseguenza è stato affrontato il problema SLAM senza utilizzare strumenti probabilistici come descritto nel seguente FSA:



L’algoritmo di esplorazione proposto è un algoritmo ricorsivo dove RescueRobot per ogni cella in cui si trova, tramite i dati provenienti dai quattro *DistanceSensor*: stabilisce lo stato delle quattro celle adiacenti nelle corrispondenti direzioni, aggiorna la mappa e successivamente si sposta nella prima cella adiacente che non sia né un ostacolo né una cella già visitata. L’esplorazione continua in questo modo fin quando il robot non è in grado di spostarsi in nessuna cella adiacente, in quel caso, si sceglie casualmente una cella a stato sconosciuto che abbia almeno una cella adiacente libera, e tramite la PathFindingUnit, si elabora un percorso verso tale cella considerando le celle a stato sconosciuto come ostacoli, infine dalla nuova posizione verrà richiamata la funzione di esplorazione.

L’algoritmo terminerà quando non ci sarà più nessuna cella a stato sconosciuto o sono inarrivabili da parte del robot e delle quali quindi diventa inutile conoscerne lo stato.

Durante l’esecuzione dell’algoritmo, RescueRobot, utilizzando *MovementUnit*, corregge la sua posizione spostandosi dalla posizione stimata reale a quella della cella in cui si trova, soggetta all’accumularsi di piccoli errori durante gli spostamenti precedenti.

In seguito sono illustrati degli esempi di output di RescueRobot durante l’esplorazione e la generazione della mappa.

```
[RESCUE_ROBOT]: MAP SITUATION: ...
 -6     -4.5    -3     -1.5     0     1.5     3     4.5     6
 (S, )  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (O, )  (F,*)  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (O, )  (F,*)  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (O, )  (F,*)  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (F, )  (F,*)  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (F, )  (F,*)  (F, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (O, )  (F,*)  (F, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (F, )  (F,*)  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
 (F,*)  (F,*)  (O, )  (S, )  (S, )  (S, )  (S, )  (S, )  (S, )
[RESCUE_ROBOT]: CORRECTING POSITION...
```

```
*** MAP EXPLORATION COMPLETED ***
-6     -4.5    -3     -1.5     0     1.5     3     4.5     6
(S, )  (O, )  (S, )  (O, )  (O, )  (F, )  (O, )  (O, )  (S, )
(O, )  (F,*)  (O, )  (F,*)  (F,*)  (F,*)  (F,*)  (F,*)  (O, )
(O, )  (F,*)  (O, )  (F,*)  (F,*)  (F,*)  (F,*)  (F,*)  (F, )
(O, )  (F,*)  (O, )  (O, )  (O, )  (O, )  (O, )  (F,*)  (F, )
(F, )  (F,*)  (O, )  (O, )  (O, )  (F,*)  (F,*)  (F,*)  (O, )
(F, )  (F,*)  (F,*)  (F,*)  (F,*)  (F,*)  (O, )  (O, )  (S, )
(O, )  (F,*)  (F,*)  (F,*)  (O, )  (O, )  (S, )  (S, )  (S, )
(F, )  (F,*)  (O, )  (F,*)  (F,*)  (F,*)  (O, )  (O, )  (O, )
(F,*)  (F,*)  (O, )  (O, )  (O, )  (F,*)  (F,*)  (F,*)  (F,*)
[RESCUE_ROBOT]: BACK TO BASE...
```

3.1.3 PathFindingUnit

La “PathFindingUnit” si occupa di elaborare in maniera razionale ed efficiente il percorso per andare da un punto iniziale a un punto finale all’interno della mappa implementando l’**algoritmo di ricerca informata A***.

L’algoritmo crea dei nodi, dell’ideale albero di ricerca, contenenti le celle della mappa e, partendo da un nodo iniziale, tra tutti i nodi adiacenti, dopo aver scartato quelli che contengono una cella con un ostacolo o a stato sconosciuto, espande sempre quello con funzione di valutazione *f(n)* minore:

$$f(n) = g(n) + h(n)$$

Dove *g(n)* fornisce il costo del cammino dal nodo iniziale al nodo *n* e *h(n)* è la funzione euristica che rappresenta il costo stimato del cammino più conveniente da *n* all’obbiettivo.

La funzione euristica scelta è la distanza euclidea tra la cella contenuta nel nodo n e la cella obbiettivo:

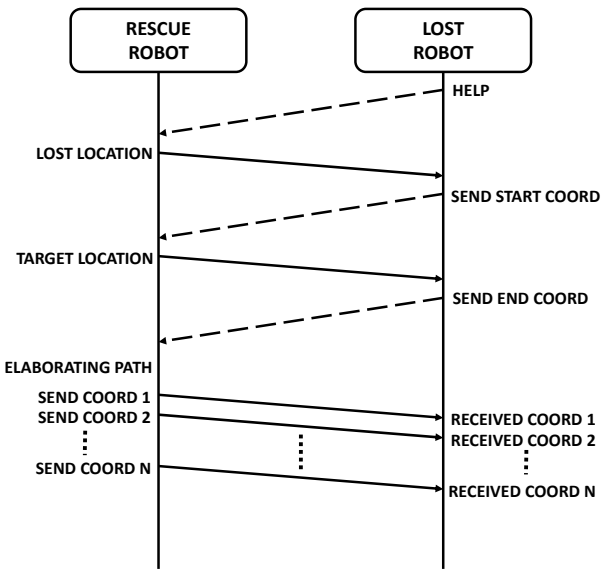
$$h(n) = \sqrt{(x_{obb} - x_{nodo})^2 + (y_{obb} - y_{nodo})^2}$$

In quanto la funzione non sbaglia mai per eccesso la stima del costo per andare da n all’obbiettivo viene definita **euristica ammissibile** e in quanto tale garantisce l’ottimalità dell’algoritmo, cioè che se dati due punti esiste una soluzione allora è sicuramente la migliore possibile.

3.1.4 CommunicationUnit

Infine *RescueRobot* fa uso della “CommunicationUnit” per interagire con i *LostRobot*.

Viene implementato un **protocollo di comunicazione** (illustrato nella figura), dove una volta finita l’esplorazione della mappa e tornato al punto “base”, *RescueRobot* si mette in ascolto nei rispettivi canali delle sue coppie emettitore/ricevitore in cerca di messaggi di aiuto da parte di un *LostRobot.* Una volta intercettato uno di questi messaggi, *RescueRobot* risponde instaurando una comunicazione di tipo **Client-Server**.



Innanzitutto il server (*RescueRobot*), richiede al client (*LostRobot*) le coordinate del punto

in cui si trova e di quello in cui deve andare, in seguito elabora un percorso con i dati di input tramite la “PathFindingUnit” e in ultima fase invia singolarmente ogni coppia di coordinate contenute nel percorso appena generato verso il client.

Si è scelto di installare in *RescueRobot* tre coppie emettitore/ricevitore per rendere contemporanea la comunicazione tra lui e tutti i *LostRobot* senza necessità di dover aspettare che si liberi un canale per instaurare una nuova comunicazione.

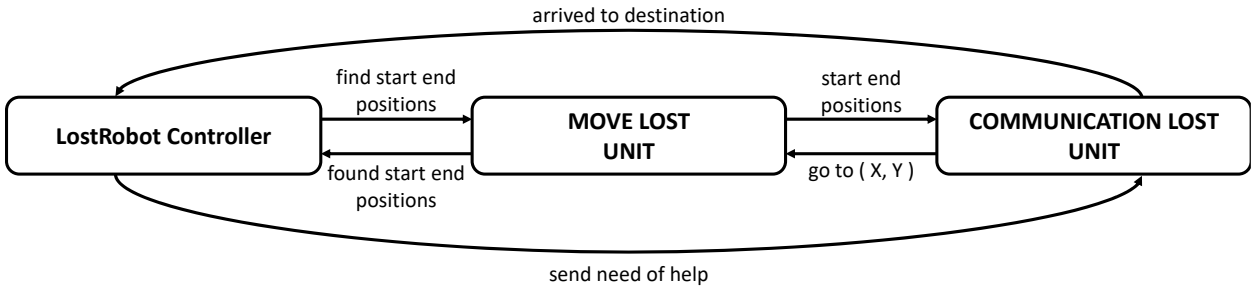
3.2 LostRobot Controller

“LostRobotController” è il controllore che gestisce i *LostRobot*, ed è unico per tutti, infatti, in fase di progettazione si è scelto di procedere in questo modo per rendere più scalabile il progetto: aldilà delle capacità computazionali della macchina su cui si esegue la simulazione, è possibile aggiungere un numero indefinito di LostRobot andando a modificare poche righe di codice in un solo controllore. In questo scenario specifico, con tre robot dispersi, il controllore tramite una funzione interna individua su quale robot è in esecuzione.

Il controllore consiste in due unità:

1. **MoveLostUnit**, si occupa della cinematica di *LostRobot* e implementa la fondamentale *collision avoidance* che verrà descritta in seguito;
2. **CommLostUnit**, unità utilizza per comunicare con *RescueRobot*.

Il comportamento generale è descritto dal seguente automa a stati finiti (FSA):



3.2.1 MoveLostUnit

La base teorica su cui si fonda la cinematica dei *LostRobot* implementata nella “MoveLostUnit” è identica a quella già illustrata nella “*MovementUnit”* del *RescueRobot* ma le due unità sono profondamente diverse in quanto funzionali a obbiettivi differenti.

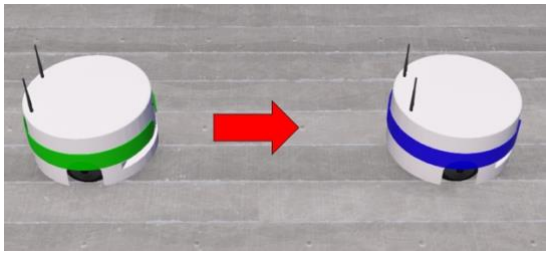
Una caratteristica fondamentale del progetto è proprio la “cecità” dei LostRobot, ovvero in grado solamente di, aldilà di comunicare, muoversi date le coordinate di due punti. Di conseguenza in un ambiente multi-agente competitivo in cui ogni robot singolarmente

tende a raggiungere il proprio obbiettivo, è quasi inevitabile che le azioni compiute da uno andranno in contrasto con quelle di qualcun altro.

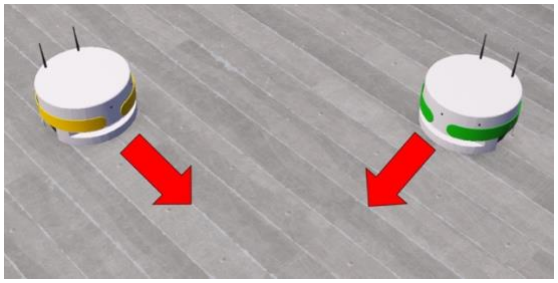
D'altro canto RescueRobot è stato progetto per fornire un percorso valido da un punto iniziale a un punto finale a ogni singolo LostRobot, senza preoccuparsi di conflitti in tali percorsi.

Date queste considerazioni è stato quindi necessario implementare un meccanismo di **collision avoidance** nei LostRobot in grado risolvere quanti più possibili scenari.

Innanzitutto, viene utilizzato il sensore di distanza frontale per verificare che la cella successiva in cui il robot deve muoversi sia libera, e in caso non lo sia attendere fin quando non lo sarà.



In seguito, ispirandosi al codice della strada, è stato inserito un sensore di priorità denominato "*prioritySensor*" che stabilisce che chi proviene da sinistra ha la priorità nell'accedere a una cella mentre l'altro rimane in attesa.



Infine, ogni LostRobot, in base al suo colore, al fine di introdurre variabilità nel comportamento, ha un proprio specifico tempo di attesa per gestire particolari situazioni di stallo.

3.2.2 CommLostUnit

L'altra unità utilizzata da *LostRobotController* è "CommLostUnit" che implementa il **protocollo di comunicazione** già descritto per il server (*RescueRobot*), dalla parte del client, quindi del robot disperso.

Nelle seguenti figure è possibile osservare l'output durante una comunicazione tra *RescueRobot* e i singoli *LostRobot*:

```
[RESCUE_ROBOT]: BACK TO BASE...
[RESCUE_ROBOT]: SWITCH CHANNEL TO LISTEN HELP REQUEST...
[RESCUE_ROBOT]: HELP CALL RECEIVED...
[RESCUE_ROBOT]: LOST SEND ME YOUR LOCATION...
[RESCUE_ROBOT]: COORDINATES RECEIVED...
[RESCUE_ROBOT]: COORDINATES RECEIVED...
[LOST_ROBOT-GREEN]: SENDING MY COORDINATES...
[RESCUE_ROBOT]: COORDINATES RECEIVED...
[RESCUE_ROBOT]: LOST ON CHANNEL 2SEND ME YOUR TARGET COORDINATES...
[RESCUE_ROBOT]: LOST ON CHANNEL 2 SEND ME YOUR TARGET COORDINATES...
[RESCUE_ROBOT]: LOST ON CHANNEL 2 SEND ME YOUR TARGET COORDINATES...
[LOST_ROBOT-GREEN]: SENDING TARGET COORDINATES...
[RESCUE_ROBOT]: LOST ON CHANNEL 2 SEND ME YOUR TARGET COORDINATES...
[RESCUE_ROBOT]: ELABORATING PATH...
[RESCUE_ROBOT]: SENDING COORDINATES...
[LOST_ROBOT-GREEN]: COORDINATE RECEIVED: 0.0 ,-4.5
[LOST_ROBOT-GREEN]: COORDINATE RECEIVED: 0.0 ,-3.0
```

```
[RESCUE_ROBOT]: LOST ON CHANNEL 3 SEND ME YOUR TARGET COORDINATES...
[RESCUE_ROBOT]: ELABORATING PATH...
[RESCUE_ROBOT]: SENDING COORDINATES...
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 3.0 ,-4.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 3.0 ,-3.0
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 4.5 ,-3.0
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 4.5 ,-1.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 4.5 ,0.0
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 3.0 ,0.0
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 1.5 ,0.0
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 1.5 ,1.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 0.0 ,1.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: -1.5 ,1.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: -1.5 ,3.0
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: -1.5 ,4.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 0.0 ,4.5
[LOST_ROBOT-BLUE]: COORDINATE RECEIVED: 1.5 ,4.5
```

In output troviamo una descrizione dettagliata di ogni singolo messaggio scambiato durante la simulazione, dove ogni messaggio è caratterizzato dal suo **mittente**, indicato tra parentesi quadre, e il **corpo** contenente l'informazione vera e propria.

4. File forniti

Nella cartella principale "*RescueLostScenario*" sono presenti le seguenti sotto cartelle, ognuna delle quale contiene ciò che è stato descritto precedentemente:

controllers

Cartella che contiene il codice Java dei due controllori con le rispettive unità utilizzate, suddivisa nei seguenti package:

- *LostRobot*
  - ***LostRobot.java***, classe che istanzia e avvia il controllore del LostRobot;
  - *Controller*
    - ***LostRobotController.java***, classe che implementa il controllore;
  - *LostUnits*
    - ***CommLostUnit.java***, classe che implementa l'unità di comunicazione;
    - ***MoveLostUnit.java***, classe che implementa l'unità di movimento e collision avoidance;

- *RescueRobot*
  - *LostBlue.wbo, LostGold.wbo, LostGreen.wbo,* file *webots object* che contengono la descrizione dei nodi da importare rappresentati i LostRobots;
  - ***RescueRobot.java***, classe che istanzia e avvia il controllore del RescueRobot;
  - *Controller*
    - ***RescueRobotController.java***, classe che implementa il controllore;
  - *Units*
    - ***MovementUnit.java***, classe che implementa l'unità di movimento;
    - ***MapExplorationUnit.java***, classe che implementa l'unità di esplorazione della mappa;
    - ***PathFinding.java***, classe che implementa l'unità di elaborazione del percorso;
    - ***CommunicationUnit.java***, classe che implementa l'unità di comunicazione;
  - *Util*
    - ***Grid.java***, classe per istanziare e gestire la mappa;
    - ***Node.java***, classe per istanziare e gestire i nodi dell'albero di ricerca;
    - ***Cell.java***, classe per istanziare e gestire le celle della mappa;

Ogni classe Java viene anche compilata e fornita con il rispettivo file *.class* per rendere i robot pronti per l'utilizzo in simulazione.

protos

Cartella che contiene i file proto dei robot utilizzati nella simulazione.

- *TiagoBaseRESCUER*, per il RescueRobot;
- *TiagoBaseLOST*, per il LostRobot blu;
- *TiagoBaseLOST2*, per il LostRobot giallo;
- *TiagoBaseLOST3*, per il LostRobot verde;

worlds

Cartella contenente il file *finalScenario.wbt* rappresentante l'ambiente di simulazione su Webots.

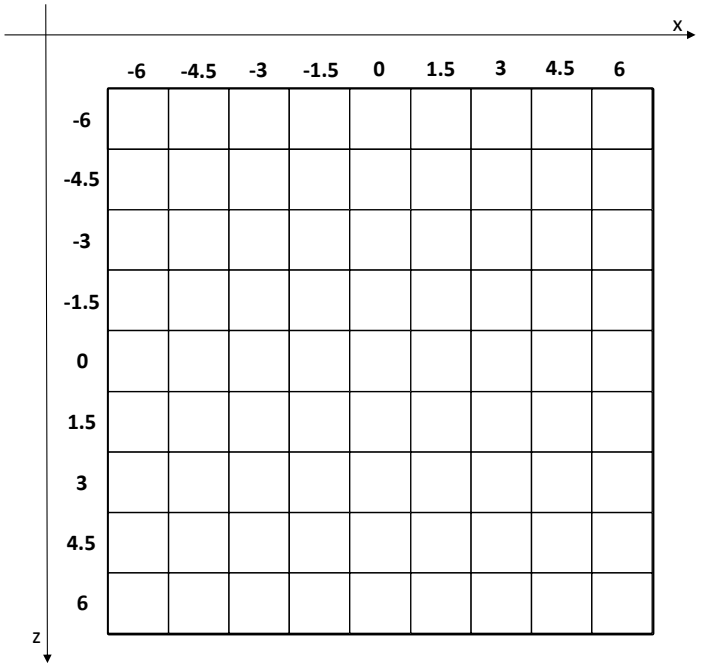
Se tutto è stato configurato correttamente, per avviare la simulazione basterà aprire il file world fornito.

Il file è stato creato con la versione **R2021b** di Webots, di conseguenza se aperto con versioni differenti potrebbero verificarsi problemi di compatibilità e quindi generarsi errori sia nel controllore sia nel caricamento dell'ambiente e degli ostacoli in esso.

Si è scelto di inserire un sistema di generazione casuale dei LostRobots in modo da poter testare più scenari possibili in breve tempo.

E' possibile modificare liberamente invece gli ostacoli presenti nell'ambiente, con l'unica limitazione di posizionarli nelle coordinate dei punti della matrice rappresentate l'ambiente, e per ovvie ragioni, cercando di non bloccare le posizioni obbiettivo dei singoli robot.

Lo schema della matrice con i rispettivi punti è illustrato in figura.



5. Conclusioni

Il progetto anche se realizzato in ambito accademico, con tutte le semplificazioni del caso, presenta dei risvolti per possibili utilizzi reali futuri.

Per esempio, in un vasto ambiente indoor di tipo “labirintico”, quale un data center o un magazzino di un azienda, in cui coesistono più robot che svolgono compiti diversi, può succedere che improvvisamente, per un motivo generico, uno di questi ha un danno all’unità di movimento-posizionamento e non è più in grado di raggiungere un punto target, che potrebbe essere il punto dove ricevere assistenza per il danno.

A questo punto l’azienda potrebbe prevedere, piuttosto che un intervento umano, la presenza di un altro robot il cui scopo è solamente quello di esplorare l’ambiente e aiutare i robot “dispersi” nel raggiungere il desiderato punto di destinazione.

Si immagini in un data center distribuito su una superficie di svariati metri quadrati quanto può essere oneroso l’intervento fisico di una persona.

In conclusione, quindi, è chiaro che il progetto realizzato è solo un punto di partenza, in cui ci si è focalizzati più sull’implementare le nozioni teoriche che sui possibili utilizzi reali e dove il passo successivo da fare sarebbe sicuramente quello di introdurre e gestire gli errori sensoriali, inevitabilmente presenti in uno scenario fisico reale.