POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

DEFINIZIONE E VALUTAZIONE DI STRATEGIE DI ESPLORAZIONE BASATE SU MCDM PER SEARCH AND RESCUE CON ROBOT MOBILI AUTONOMI

Laboratorio di Robotica e Intelligenza Artificiale
del Politecnico di Milano

Relatore: prof. Francesco AMIGONI
Correlatore: ing. Nicola BASILICO

Tesi di Laurea Magistrale di:
Alessandra TONETTI, matricola 682302
Dario FACCHI, matricola 682303

Anno Accademico 2009-2010

A te, mamma, che mi ha hai sempre aiutato, sostenuto e hai sempre creduto in me, anche quando io ero la prima a non farlo...
Ale

Ai mie genitori, che mi sono sempre stati vicino, dandomi la possibilità di intraprendere questo percorso.
Dario

I computer sono incredibilmente veloci, accurati e stupidi.
Gli uomini sono incredibilmente lenti, inaccurati e intelligenti.
L'insieme dei due costituisce una forza incalcolabile.

— Albert Einstein

Sommario

Durante le operazioni di soccorso in ambienti disastrati, squadre di robot possono essere impiegate per rilevare informazioni importanti, tra cui la presenza di ostacoli e vittime, in luoghi che non possono essere raggiunti facilmente e senza rischi da operatori umani. Per permettere ai robot di determinare le posizioni da raggiungere nell'ambiente è necessario sviluppare una strategia di esplorazione che, ad ogni passo, selezioni il migliore candidato in un insieme di punti raggiungibili, valutati in base a diversi criteri. In questa tesi proponiamo lo sviluppo di una strategia di esplorazione per robot autonomi impiegati in operazioni di soccorso basata su Multi-Criteria Decision Making (MCDM), un metodo che permette di aggregare i criteri all'interno di una funzione di valutazione considerando le relazioni di dipendenza tra essi. Questo approccio non dipende dal numero e dalla natura dei criteri considerati ed è flessibile in quanto permette di aggiungere facilmente altri criteri per la valutazione di un punto. Dopo l'implementazione della strategia abbiamo effettuato una serie di esperimenti ottenendo risultati soddisfacenti e dimostrando l'efficacia dell'impiego di Multi-Criteria Decision Making in applicazioni di ricerca e soccorso.

Ringraziamenti

Vogliamo innanzitutto ringraziare il Professor Francesco Amigoni per la sua disponibilità e per averci dato la possibilità di realizzare questo progetto inserito in un campo per noi molto interessante.
Ringraziamo, inoltre, l'ingegnere Nicola Basilico che ci ha seguito per tutto lo svolgimento della tesi, per i suoi consigli e per averci aiutato nelle fasi critiche del lavoro.
Infine un grazie di cuore a tutti gli amici che ci hanno sopportato e sostenuto durante questo nostro percorso.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Durante le operazioni di soccorso in ambienti disastrati, gli operatori umani possono essere aiutati da squadre di robot, per raccogliere informazioni in luoghi difficili e rischiosi da raggiungere. In questo ambito opera la RoboCup Rescue, una competizione internazionale che promuove ricerca e sviluppo in questo campo e di cui fa parte, dal 2006, la Virtual Robot Competition, dove squadre di robot virtuali hanno il compito di esplorare ambienti simulati, inizialmente sconosciuti, al fine di trovare le vittime presenti e costruire una mappa abbastanza dettagliata con l'indicazione della loro posizione. Uno degli aspetti più importanti che devono considerare i ricercatori partecipanti a questa competizione riguarda lo sviluppo di una strategia di esplorazione efficiente che permetta agli agenti di determinare il prossimo punto da raggiungere all'interno di un ambiente parzialmente esplorato. Per selezionare questa posizione vengono valutati dei punti candidati, solitamente aggregando diversi criteri all'interno di una funzione di utilità [6, 20], come è stato fatto anche da alcuni partecipanti alla RoboCup Rescue Virtual Robot Competition [23, 28].

In questa tesi proponiamo lo sviluppo di una strategia di esplorazione per applicazioni di soccorso basata su Multi-Criteria Decision Making (MCDM) [4], un metodo che permette di scegliere tra un insieme di alternative sulla base di una composizione di diversi criteri. MCDM combina i criteri permettendo di assegnare ad ognuno un peso che ne identifica l'importanza, ma anche di

attribuire un peso agli insiemi composti da più criteri in modo da definire relazioni di ridondanza e sinergia tra essi. Molti degli approcci alternativi si basano su funzioni fortemente dipendenti dal numero e dal tipo di criteri che aggregano, rendendo il loro utilizzo difficile in diversi contesti. MCDM, invece, è un metodo molto flessibile che è già stato impiegato in diversi campi tra cui, oltre l'informatica, anche l'economia e l'ecologia. È indipendente dal numero e dalla natura dei criteri considerati e permette di aggiungerne di nuovi all'interno della funzione di valutazione di un candidato in modo molto semplice, senza dover studiare dal principio una nuova funzione, ma semplicemente intervenendo sui pesi assegnati ai criteri e agli insiemi composti da tutte le loro possibili combinazioni.
Partendo da un software esistente, sviluppato nel 2009 dai ricercatori delle università di Amsterdam e di Oxford per partecipare alla RoboCup Rescue Virtual Robot Competition, abbiamo potuto implementare la nostra strategia senza dover sviluppare algoritmi di mapping, localizzazione, navigazione, ecc... Nella versione principale del nostro codice, abbiamo considerato tre criteri nella funzione di aggregazione. Questi, considerando un determinato robot, facente parte di una squadra, e una determinata frontiera, valutano la distanza che li separa, la quantità di area stimata che è possibile esplorare al di là della frontiera in esame e con quale probabilità il robot sarà ancora in grado di comunicare con gli altri una volta raggiunta tale frontiera. Oltre a questa prima versione, spinti da diverse motivazioni ne abbiamo sviluppate altre due. La prima è stata creata per risolvere un problema riscontrato durante gli esperimenti e dovuto al software originale, mentre l'altra è stata sviluppata allo scopo di dimostrare la flessibilità di MCDM. Per fare ciò abbiamo considerato nel calcolo dell'utilità una nuova informazione data dalla quantità di carica della batteria necessaria ad un robot per raggiungere una determinata frontiera.
Per poter valutare la validità del nostro approccio abbiamo effettuato una serie di esperimenti simulati in due ambienti con diverse caratteristiche, confrontando le prestazioni del nostro metodo con quelle della strategia presente nel software originale, su cui ci siamo basati nel nostro lavoro, e con quelle di altre due strategie proposte in letteratura, ottenendo risultati soddisfacenti.

Altre prove sono state fatte per dimostrare di aver risolto il problema riscontrato e che ci ha portato allo sviluppo della seconda versione e per verificare le prestazioni dopo l'inserimento della nuova informazione nel calcolo dell'utilità.
La tesi è strutturata nel modo seguente. Nel Capitolo 2 sono presentati la RoboCup Rescue, il simulatore utilizzato durante la Virtual Robot Competition e nei nostri esperimenti e i lavori correlati riguardanti le strategie di esplorazione. Nel Capitolo 3 si illustra nel dettaglio l'approccio MCDM. Nel Capitolo 4 è descritto il funzionamento del sistema e come sono state sviluppate le diverse versioni del software, nonché le altre strategie per il confronto delle prestazioni. Nel Capitolo 5 sono mostrati i risultati sperimentali, ottenuti dall'esecuzione di duecento prove, e la loro valutazione. Nel Capitolo 6 si riassumono lo scopo del lavoro, cosa è stato fatto, le valutazioni e le prospettive future. Nell'Appendice A si riportano le istruzioni per l'installazione e il funzionamento del software. Nell'Appendice B viene mostrato un esempio di utilizzo del sistema. Nell'Appendice C sono raccolte tutte le posizioni iniziali dei robot utilizzate negli esperimenti effettuati.

Capitolo 2

Stato dell'arte

2.1 RoboCup Rescue

La *RoboCup Rescue*[1] [12] è un'importante competizione inserita nell'ambito della *RoboCup*[2], un'iniziativa internazionale che cerca di incoraggiare la ricerca nel campo dell'intelligenza artificiale e della robotica intelligente. In particolare la RoboCup Rescue mira a promuovere ricerca e sviluppo di tecnologie per il salvataggio di esseri umani in ambienti colpiti da catastrofi quali terremoti, maremoti, tornado, incidenti aerei e ferroviari e così via. La RoboCup Rescue è divisa in due aree distinte:

- *Rescue Robot League*, che consiste nella creazione di robot reali che devono trovare vittime muovendosi all'interno di ambienti non strutturati. Le sfide principali affrontate dai partecipanti a questa competizione spaziano dalla meccatronica per la locomozione avanzata alla percezione e pianificazione per fornire piena autonomia al robot. Il comportamento degli agenti viene valutato all'interno di speciali arene di test, ricreate in laboratorio e sviluppate dal National Institute of Standards and Technology[3] (NIST), che presentano diversi livelli di difficoltà così da fornire un trampolino di lancio per il mondo reale.

[1] http://www.robocuprescue.org/
[2] http://www.robocup.org/
[3] http://www.isd.mel.nist.gov/projects/USAR/

- *Rescue Simulation League*, è un progetto diviso, a sua volta, in due parti principali ma strettamente connesse tra loro:

  - *Agent Simulation Competition*, che mira a sviluppare simulatori realistici che consentono di emulare i principali fenomeni in caso di catastrofi (simulatore del traffico, del fuoco, ecc...) e agenti che abbiano la capacità di agire in uno scenario di questo genere.
  - *Virtual Robot Competition*, che è il punto d'incontro per i ricercatori coinvolti nella Agent Simulation Competition e nella Rescue Robot League. Si tratta di una simulazione dei robot reali utilizzati nella Rescue Robot League, che permette di concentrarsi sugli aspetti di alto livello della programmazione dei robot, di eliminare il bisogno di hardware specifico, con conseguente abbassamento dei costi, e di eseguire esperimenti in arene simulate molto più grandi di quelle riprodotte in laboratorio. Grazie a questo, inoltre, è possibile utilizzare squadre di robot più numerose. La Virtual Robot Competition si basa su *USARSim*, un simulatore, di cui parleremo nella Sezione 2.2, che permette di ricreare ambienti e agenti con le stesse caratteristiche di quelli reali.

Il settore della RoboCup Rescue che abbiamo preso in considerazione nell'affrontare questo progetto è quello relativo alla Virtual Robot Competition e gli aspetti più importanti in questo campo, che devono necessariamente coesistere all'interno di un software creato per partecipare a tale competizione, si distinguono in:

- Esplorazione, che consiste nello sviluppo di una strategia che permetta agli agenti di muoversi nell'ambiente in modo efficiente scegliendo ad ogni passo un nuovo obiettivo da raggiungere.

- Localizzazione e mapping, che costituiscono il processo di *Simultaneous Localization And Mapping* (SLAM) [8], il quale, a partire dai rilevamenti dei sensori, si occupa di determinare e tracciare la posizione di ogni robot e di ricostruire una mappa raffigurante l'ambiente esplorato.

- Pianificazione del percorso o *Path Planning*, che effettua la ricerca del miglior tragitto percorribile da un determinato robot per raggiungere l'obiettivo definito nella fase di esplorazione.
- Navigazione, che permette ad ogni agente di muoversi dalla posizione in cui si trova verso la sua destinazione seguendo il percorso definito tramite Path Planning.
- Riconoscimento delle vittime, che è un aspetto molto importante, nel contesto in esame, per il quale vengono utilizzati dei sensori, creati appositamente, che sono in grado di riconoscere parti del corpo umano. Questi permettono, ai robot che ne fanno uso, di identificare le vittime e di distinguerle da altri ostacoli presenti nell'ambiente. Per ogni vittima deve essere indicata sulla mappa la posizione in cui si trova in modo da permettere ai soccorsi di intervenire il più velocemente possibile.

2.2 USARSim

USARSim (Unified System for Automation and Robot Simulation) [24] è un simulatore 3D ad alta fedeltà, orientato al soccorso robotico, nato come strumento di ricerca all'interno di un progetto della National Science Foundation (NSF), riguardante lo studio di robot, agenti e squadre di operatori nel contesto di *Urban Search And Rescue* (USAR, operazioni di ricerca e soccorso in ambiente urbano). Il simulatore sfrutta il motore grafico del gioco Unreal Tournament 2004, UnrealEngine2[4], un software commerciale multipiattaforma creato per sviluppare giochi sparatutto multigiocatore in prima persona, ideato dalla Epic Games, che grazie ad una fisica avanzata, un dettaglio grafico abbastanza alto e la personalizzazione degli scenari è l'ambiente perfetto per poter sfruttare al meglio le potenzialità di USARSim. Lasciando quindi gestire la parte più complessa della simulazione al motore grafico 3D di Unreal, il programmatore può dedicarsi alla creazione di sensori, attuatori, sistemi di controllo e strumenti d'interfaccia in tutta libertà, rendendo il robot simulato simile a quello reale o sviluppandone di nuovi senza aver bisogno

[4] http://www.unrealtechnology.com/

di parti meccaniche o fisiche.
USARSim è un programma gratuito e, soprattutto, open-source, questo vuol dire che chiunque può migliorarlo creando robot, sensori, attuatori e mappe, grazie anche all'utilizzo di software, integrati nel motore di gioco, come l'editor grafico Unreal Editor ed il linguaggio di scripting interno Unreal Script, in modo da poter condividere il proprio lavoro con altri, espandendo così il simulatore con nuovi modelli di cui chiunque può servirsi.
Gli utilizzatori di USARSim, come ad esempio le squadre partecipanti alla RoboCup Rescue Virtual Robot Competition, dopo aver sviluppato per i propri robot un algoritmo di mapping e localizzazione, una strategia di movimento e tutto ciò che riguarda l'intelligenza dell'agente, possono posizionarli nell'ambiente virtuale per testarne il funzionamento. Un grande vantaggio derivante dall'utilizzo di USARSim è dato dal fatto che il costo dei test effettuati in simulazione è nettamente inferiore rispetto all'utilizzo di robot in ambienti reali creati *ad hoc*.

2.3 Lavori correlati

Trovare la migliore strategia di movimento per robot che si muovono all'interno di un ambiente sconosciuto, nonché luogo di un disastro, con lo scopo di costruirne una mappa abbastanza accurata e di individuare le vittime presenti è uno dei principali problemi affrontati dalle squadre che partecipano alla RoboCup Rescue Virtual Robot Competition.
Per valutare i risultati raggiunti dalle squadre iscritte alla gara viene utilizzata una formula che considera fattori riguardanti le vittime (numero e stato), la mappa creata (qualità e accuratezza) e l'esplorazione (quantità di m^2 osservati). Vengono inoltre considerate penalità le collisioni e gli interventi sui robot, durante la navigazione.
Diverse sono le strategie adottate fino ad oggi e, in base a queste, i robot sul posto possono essere comandati da un operatore umano oppure avere un comportamento del tutto autonomo. Nel caso di robot telecomandati è l'operatore umano che decide il comportamento del robot. Tale strategia è stata adottata dalla squadra della Carnegie Mellon University di Pittsburgh,

Steel, che alla prima RoboCup Rescue Virtual Robot Competition tenutasi a Brema nel 2006, a causa dell'utilizzo dell'operatore, è stata penalizzata nella classifica finale, ma ha vinto il premio speciale come miglior interfaccia uomo-macchina [2]. Steel, che si basa sull'idea che robot e umani debbano lavorare insieme per raggiungere l'obiettivo, ha utilizzato un Multirobot Control System (MrCS) [21] per permettere ai robot di navigare autonomamente e in modo cooperativo ed un proxy, chiamato Machinetta [16], implementato in Java e disponibile gratuitamente in internet, che prende la maggior parte delle decisioni, mentre viene assegnato ad un operatore umano il compito di effettuare le scelte fondamentali. L'operatore si connette al proxy attraverso un'interfaccia e può intervenire sulla squadra di robot su tre livelli. Al livello più basso, l'operatore sceglie un robot in modo da assumerne il controllo. Al livello intermedio interagisce con il robot modificando il suo piano di esplorazione. Al più alto livello può intervenire sull'intera squadra di robot modificando le priorità delle zone che devono ancora essere visitate influendo, tramite questo assegnamento, sull'esplorazione effettuata dai robot. L'uomo ha sempre la più alta autorità, anche se il robot può modificare leggermente il suo percorso per evitare ostacoli o posizioni pericolose.
La maggior parte delle squadre, tuttavia, implementa comportamenti autonomi per i propri robot. Una di queste è la squadra della International University di Brema (oggi nota come Jacobs University), *Virtual IUB*, che ha partecipato alla RoboCup Rescue Virtual Robot Competition del 2006 ottenendo il secondo posto [2]. Durante la missione di soccorso ogni robot agisce individualmente, senza alcun coordinamento con gli altri, la cooperazione si verifica solo quando la missione è finita e i robot uniscono i loro risultati parziali individuali. Il maggior sforzo della squadra Virtual IUB è stata l'integrazione di sistema, piuttosto che lo sviluppo di nuove tecniche algoritmiche. Il software disponibile esistente è stato riutilizzato ogni volta in cui è stato possibile. I quattro compiti fondamentali (navigazione, mapping, esplorazione e rilevamento vittime) sono stati implementati in quattro moduli separati ed eseguiti in parallelo. La strategia di esplorazione, che è ciò di cui ci interessiamo in questo progetto ed è il componente principale di tutto il sistema, utilizza l'idea dell'esplorazione basata su frontiere [25].

Le frontiere sono regioni al confine tra spazio aperto e spazio inesplorato e muovendosi verso nuove frontiere il robot può estendere la propria mappa finché l'intero ambiente non è stato visitato. Ad una frequenza fissata si ottiene un'istantanea della mappa più recente prodotta dal mapping. Si tratta di una mappa a griglia in cui ogni cella viene associata ad uno stato che può essere libero, occupato o sconosciuto. Si estraggono tutte le frontiere disponibili dalla mappa attuale (una cella si dice di frontiera se è libera ed è adiacente ad una cella sconosciuta, di conseguenza una frontiera è un insieme di celle di frontiera contigue) dopodiché viene determinata quella più vicina al robot e viene chiamata una funzione di navigazione verso la frontiera scelta. Se durante il moto verso la frontiera selezionata viene rilevata una vittima, si attiva una nuova funzione di navigazione che ha come obiettivo la vittima rilevata, piuttosto che la frontiera identificata precedentemente.
La squadra dell'università di Friburgo, *Rescue Robots Freiburg* [13], ha vinto la RoboCup Rescue Virtual Robot Competition del 2006 [2]. L'idea chiave alla radice di questo sistema è che i robot pianificano percorsi ed esplorano l'ambiente basandosi su una vista locale dello stesso, che rimane consistente grazie all'utilizzo di una comunicazione indiretta fatta tramite tag RFID [28]. I tag RFID forniscono un numero unico che può essere letto fino ad un metro di distanza. La localizzazione degli agenti, fatta attraverso il rilevamento di questi tag, è computazionalmente meno costosa e più precisa rispetto alla localizzazione fatta attraverso le immagini della videocamera o tramite i rilevamenti dei sensori. Inoltre è possibile memorizzare dati, per esempio riguardanti stanze vicine o vittime, direttamente nei tag. Durante la navigazione ogni robot mantiene in memoria un Local RFID Set (LRS) che contiene tutti gli RFID percepiti entro un certo raggio dalla sua posizione. In base a questa informazione i robot rilasciano nuovi RFID nell'ambiente in modo da mantenerne una densità predefinita. L'esplorazione inizia con l'estrazione di un insieme $F = \{f_j\}$ contenente tutte le frontiere [25], e per ogni posizione candidata $f_j \in F$ viene calcolato un valore di utilità tramite la formula:

$$u(f_j) = -\gamma_1 \cdot angle(f_j) - \gamma_2 \cdot visited(f_j)$$

Il termine $angle(f_j)$ fa in modo che il robot preferisca frontiere posizionate davanti a sé e fa sì che il robot non cambi direzione troppo spesso. È un valore che cresce con l'angolo compreso tra la frontiera da raggiungere e la direzione verso cui è diretto il robot. Durante l'esplorazione ogni membro della squadra memorizza nel più vicino RFID la propria posizione p a cui viene associato un valore $count(p)$, che viene incrementato dai robot ogniqualvolta la posizione p viene raggiunta. Questa quantità, insieme a $d(f_j, p)$, ossia la misura della distanza tra la frontiera f_j e la posizione p, viene utilizzata per determinare $visited(f_j)$, che serve al robot per ricordare quali sono le aree già esplorate e che viene calcolato secondo la formula:

$$\sum_{r \in LRS} \sum_{p \in P_r} (1/d(f_j, p)) \cdot count(p)$$

Ad ognuno dei due termini appena descritti viene associato un peso, rappresentato rispettivamente da $\gamma 1$ e $\gamma 2$. I robot, quindi, utilizzano gli RFID non solo per mantenere memoria del loro passato ma anche di quello degli altri, implementando una forma di comunicazione indiretta.
Una delle strategie più interessanti è quella implementata dalla squadra dell'università di Amsterdam, *UvA Rescue*, che alla RoboCup Rescue Virtual Robot Competition del 2006 si è aggiudicata il terzo posto ed ha vinto il premio per il miglior algoritmo di mapping [2], mentre nella competizione tenutasi ad Atlanta nel 2007 ha raggiunto le semifinali. Dal 2008, insieme alla squadra di Amsterdam, ha partecipato allo sviluppo del sistema anche l'università di Oxford e il nome della squadra è stato cambiato in *Amsterdam Oxford Joint Rescue Forces* (AOJRF) [22], la quale ha ottenuto molti risultati importanti tra cui il primo posto alla RoboCup Rescue Virtual Robot Competition del 2008 in Brasile. Inizialmente il compito fondamentale assegnato ad ogni robot era quello di cercare le vittime all'interno dell'ambiente da esplorare mentre le informazioni da inserire nella mappa venivano acquisite passivamente durante il perseguimento di questo obiettivo [18]. Nelle versioni successive del software è stata, invece, adottata un'esplorazione attiva il cui obiettivo principale è quello di aumentare al massimo la conoscenza dell'ambiente e la scoperta di vittime viene ritenuta un effetto collaterale

di un'esplorazione efficiente. Le frontiere estratte durante l'esplorazione sono definite come il confine tra la regione sicura (l'ambiente "visto" da un sensore laser con un range di circa 3 metri) e lo spazio libero (l'ambiente "visto" da un sensore laser con un range di circa 20 metri). Ad ogni frontiera presente in un dato istante viene assegnato un valore di utilità tramite l'equazione:

$$u(f) = A(f) \cdot P(f)/d(f)$$

dove $A(f)$, $P(f)$ e $d(f)$ rappresentano rispettivamente la stima dell'area presente oltre la frontiera f, la probabilità che il robot possa ancora comunicare con la ComStation, un dispositivo fisicamente presente nell'ambiente che raccoglie le informazioni riguardanti la mappa inviategli dai robot, una volta raggiunta tale frontiera e la distanza tra questa e il robot [23]. Nella Sezione 4.1 parleremo ampiamente di questo sistema dato che è quello su cui si basa tutto il nostro software.
Un gruppo dell'università La Sapienza di Roma propone, al contrario, un approccio che non si basa sull'ottimizzazione di una funzione bensì sull'implementazione di un componente responsabile delle attività di monitoraggio e guida del processo di esplorazione e ricerca [7], modellato usando PNP (Petri Net Plans) [27], un formalismo basato sulle Reti di Petri [14]. PNP è un meccanismo altamente flessibile per realizzare il livello decisionale del robot e fornisce un metodo per decidere quando riconsiderare le decisioni e come scegliere tra diversi obiettivi. Questo è utile in quanto molto spesso, quando i compiti di esplorazione e ricerca vengono eseguiti simultaneamente, è necessario interrompere l'azione che si sta eseguendo se viene rilevato un obiettivo più importante durante la navigazione. Un esempio di strategia sviluppata con questo formalismo è mostrato nella Figura 2.1. Secondo tale strategia, chiamata *Victims First*, l'analisi delle vittime è preferita all'esplorazione dell'ambiente. Se come obiettivo si sceglie di raggiungere una possibile vittima viene eseguita l'azione `navigateToCandidateVictim`, altrimenti si attua il comportamento `navigateAndSearch` che può essere interrotto, nel caso in cui venga vista una nuova possibile vittima, dalla transizione `newVictimFound` usata per disattivare `navigateAndSearch` e attivare

`navigateToCandidateVictim`.

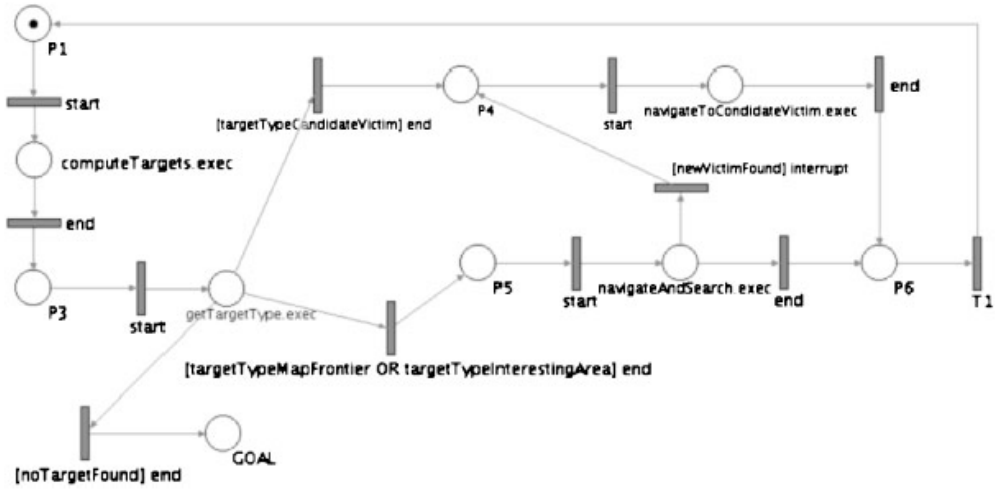


Figura 2.1: Strategia Victims First

La Figura 2.2 mostra in dettaglio come agisce `navigateToCandidateVictim`.

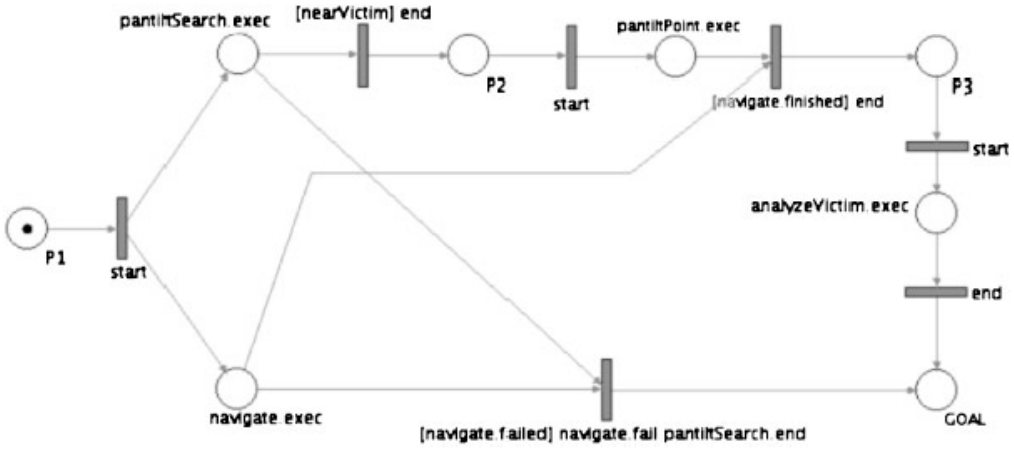


Figura 2.2: `navigateToCandidateVictim`

Gli esempi appena citati descrivono nel dettaglio strategie create appositamente per la partecipazione alla RoboCup Rescue Virtual Robot Competition, ma è necessario sottolineare che il problema generale della definizione

di strategie di esplorazione, non strettamente legate alla RoboCup Rescue, è stato affrontato anche da diversi lavori in letteratura. L'approccio tradizionale, chiamato *Next-Best-View* (NBV), che sarà discusso nel Capitolo 3 insieme al metodo, basato su questa tecnica, che abbiamo utilizzato per implementare la nostra strategia di esplorazione, consiste nello scegliere la prossima posizione da raggiungere tra un insieme di candidate, valutandole in base ad alcuni criteri. Generalmente le posizioni candidate si trovano sul confine tra lo spazio aperto conosciuto e le aree non ancora esplorate, ma la loro definizione può variare, come vedremo nel Capitolo 4 parlando delle frontiere considerate in questo progetto. Per effettuare la valutazione delle frontiere si possono utilizzare diversi tipi di funzioni. Un esempio di calcolo dell'utilità relativa ad una posizione candidata p è dato da:

$$u(p) = A(p) \cdot e^{-\lambda \cdot c(p)}$$

In questa formula vengono considerati due criteri, $c(p)$ e $A(p)$, che definiscono rispettivamente il costo necessario per raggiungere p e l'area potenzialmente osservabile da tale posizione, e viene utilizzata la variabile λ che rappresenta un peso usato per bilanciare i due criteri [10]. Ad eccezione di formule simili a quella appena menzionata [5, 6], si distinguono anche tecniche basate sull'entropia relativa [1] e altre in cui il criterio che misura il costo è linearmente combinato con la riduzione dell'incertezza della mappa, prevista dopo un'osservazione [19]. In alcuni metodi, invece, vengono considerati molti criteri diversi tra loro, tra cui anche il numero di caratteristiche visibili nell'ambiente, le rotazioni e il numero di stop necessari al robot per seguire il percorso verso la frontiera considerata, inseriti in una complessa funzione che li comprende tutti, per valutare la posizione candidata [20]. Esistono, inoltre, strategie studiate appositamente per gestire situazioni in cui l'esplorazione è eseguita da più di un robot e che valutano le frontiere per mezzo di una funzione che considera, oltre al costo, un criterio, inizialmente posto ad un valore uguale per ogni agente, che decresce con la vicinanza degli altri robot [26].

Capitolo 3

Multi-Criteria Decision Making

Nell'ambito della RoboCup Rescue Virtual Robot Competition i robot autonomi che compiono attività di esplorazione e ricerca all'interno di mappe virtuali, che simulano zone danneggiate da disastri ambientali o causati da errori umani, devono saper eseguire diverse funzioni contemporaneamente. Ogni agente deve, perciò, essere in grado di localizzare se stesso all'interno dell'ambiente in cui si trova ma anche di costruirne una mappa abbastanza dettagliata che, in una vera situazione di pericolo, possa essere compresa da chi presterà soccorso alle vittime coinvolte. I robot, inoltre, devono saper decidere quale sarà la prossima meta da raggiungere per esplorare in modo ottimale l'intero ambiente. Per fare ciò è necessario lo sviluppo di una strategia di esplorazione.
L'obiettivo di questo lavoro è proprio quello di studiare e di implementare una buona strategia di esplorazione per il tipo di agenti appena descritto, in modo che questi visitino più area possibile e, di conseguenza, rendano disponibile, a lavoro ultimato, una mappa che comprenda la maggior parte del territorio esaminato con l'indicazione di ostacoli e vittime al suo interno. Non disponendo a priori di nessuna conoscenza riguardo le posizioni delle vittime coinvolte, nell'ambito della RoboCup Rescue, massimizzare l'area esplorata equivale a massimizzare il numero di vittime trovate dai robot.
Documentandoci sul lavoro svolto fino ad oggi abbiamo notato che la maggior parte delle persone attive in questo campo ha progettato strategie basate su

formule studiate *ad hoc*, di cui fanno parte quelle discusse nella sezione 2.3. Queste dipendono fortemente dal numero e dal tipo di criteri considerati e, anche se la loro applicazione nel contesto per cui sono state create produce risultati soddisfacenti, il loro impiego in un ambito differente da quest'ultimo sarebbe difficile e non si otterrebbero le stesse prestazioni. Inoltre tali formule sono poco flessibili e anche solo inserendo un nuovo criterio sarebbe necessario generare una nuova espressione per prenderlo in considerazione. Noi, per tutti questi motivi, abbiamo deciso di adottare una strategia basata su ***Multi-Criteria Decision Making*** **(MCDM)** [4], sfruttando le potenzialità di un metodo decisionale generale e già esistente. MCDM è già stato utilizzato in diversi campi, tra cui l'economia, l'ecologia e l'informatica, e, grazie alla sua flessibilità, permette di inserire facilmente nuovi criteri all'interno della formula per il calcolo dell'utilità e di adattare quest'ultima a diverse situazioni semplicemente associando diversi pesi ai criteri presi in considerazione. Il nostro obiettivo non consiste nel cercare la migliore strategia di esplorazione, ma cerchiamo di ottenere risultati almeno paragonabili a quelli prodotti dall'applicazione delle strategie esistenti, sfruttando un metodo flessibile che ci fornisce una tecnica per definire strategie di esplorazione efficaci e adatte alla situazione presa in esame.
Le strategie basate su MCDM seguono l'approccio NBV che consiste nel mandare il robot verso una posizione, scelta tra un insieme di candidate, ad ognuna delle quali viene associato un valore, che ne misura l'utilità, calcolato considerando diversi criteri. Con queste strategie le decisioni vengono prese solo quando l'obiettivo che si sta perseguendo viene raggiunto. L'algoritmo NBV, come mostrato nella Figura 3.1, è costituito dai seguenti quattro passi principali ripetuti in sequenza:

- Acquisizione di informazioni relative all'ambiente circostante usate per costruire una mappa parziale.

- Integrazione delle informazioni, rilevate al punto precedente, nella mappa globale.

- Selezione della prossima posizione da raggiungere.

- Navigazione verso la posizione scelta.

Figura 3.1: Approccio Next-Best-View

L'approccio MCDM, in particolare, permette di calcolare un valore di utilità per ogni possibile posizione raggiungibile dal robot, ossia per ogni frontiera presente sulla mappa nell'istante dell'assegnamento. Queste frontiere sono definite come il confine tra la regione sicura e lo spazio libero e il loro processo di estrazione verrà discusso esaurientemente nella Sezione 4.1.2.
La funzione di utilità deve considerare alcuni criteri. Nel nostro caso, quelli che abbiamo analizzato nello sviluppo del sistema in esame, sono i seguenti:

- Stima dell'area che il robot potrà vedere una volta giunto alla frontiera di cui si sta calcolando l'utilità.
- Distanza tra il robot e il punto che si sta considerando.
- Probabilità che il robot possa ancora comunicare con la ComStation una volta arrivato nella posizione prestabilita.
- Quantità di batteria necessaria al robot per raggiungere l'obiettivo.

La valutazione di questi criteri, data una frontiera, avviene in maniera molto semplice e, per ottenere un buon valore di utilità, quelli che misurano la stima dell'area e la probabilità di comunicazione devono essere massimizzati mentre quelli relativi alla distanza del robot dalla frontiera e al consumo di batteria devono essere minimizzati.
MCDM si avvale di una particolare tecnica di aggregazione per i criteri considerati che consiste nel calcolare l'utilità di ogni frontiera utilizzando l'integrale di Choquet rispetto alla misura fuzzy μ [11].
Assumendo N come l'insieme degli n criteri considerati e L come l'insieme di tutte le frontiere, la formula di cui ci siamo serviti per determinare $u(p)$, ovvero l'utilità di ogni possibile posizione $p \in L$, è la seguente:

$$u(p) = \sum_{j=1}^{n} (u_{(j)}(p) - u_{(j-1)}(p)) \cdot \mu(A_{(j)})$$

Per poter calcolare $u(p)$ è necessario conoscere i valori delle singole utilità $u_j(p)$ determinate per ogni coppia $p \in L$ e $j \in N$. Dati un criterio j e una posizione p il valore $u_j(p)$, compreso nell'intervallo $[0, 1]$, è definito tramite la normalizzazione. Questa viene usata per poter valutare su una scala comune l'utilità di p in base ad ogni criterio ed è fatta considerando tutti i valori di j per ogni candidato presente nel passo di esplorazione corrente. La funzione di normalizzazione cambia in base alla natura del criterio in esame. Nell'ambito del nostro lavoro, per un criterio come la stima dell'area presente al di là della frontiera p, per cui è preferito un valore elevato, viene utilizzata la seguente formula di normalizzazione:

$$u_a(p) = (a(p) - \min_{q \in L} a(q)) / (\max_{q \in L} a(q) - \min_{q \in L} a(q))$$

Al contrario, un criterio come la distanza tra il robot e la frontiera p, per cui è preferito un valore piccolo, viene normalizzato tramite la seguente funzione:

$$u_d(p) = 1 - ((d(p) - \min_{q \in L} d(q)) / (\max_{q \in L} d(q) - \min_{q \in L} d(q)))$$

In queste formule $a(p)$, $a(q)$ $d(p)$ e $d(q)$ con $p, q \in L$ rappresentano rispettivamente i valori assunti dai criteri area e distanza nelle posizioni p e q. Una volta calcolate, le utilità relative ad ogni criterio per una data posizione devono essere ordinate in modo da avere:

$$u_{(0)}(p) = 0 \leq u_{(1)}(p) \leq \ldots \leq u_{(n)}(p) \leq 1$$

Il pedice $_{(j)}$, nella formula per calcolare $u(p)$, indica proprio questo ordinamento, mentre l'insieme definito da:

$$A_{(j)} = \{i \in N | u_{(j)}(p) \leq u_{(i)}(p) \leq u_{(n)}(p)\}$$

è quello che contiene tutti i criteri la cui utilità è compresa tra quella del j-esimo e quella del n-esimo criterio, poste in ordine crescente.
Ad ogni elemento contenuto nell'insieme delle parti di N, $\mathcal{P}(N)$, bisogna associare un valore, compreso nell'intervallo $[0, 1]$, che ne indica l'importanza. La misura fuzzy μ, introdotta in precedenza, ci permette di farlo e, quindi, di attribuire un peso non solo ad ogni singolo criterio, ma anche ad ogni loro possibile combinazione. Durante l'assegnamento dei pesi è necessario soddisfare le seguenti proprietà:

- $\mu(\emptyset) = 0$.
- $\mu(N) = 1$.
- Se $A \subset B \subset N$ allora $\mu(A) \leq \mu(B)$ dove $A \in \mathcal{P}(N)$ e $\mu(A)$ rappresenta il peso assegnato all'insieme di criteri A.

Tramite μ è possibile definire delle relazioni di dipendenza fra criteri. Dati due criteri c_1 e c_2 e i loro pesi $\mu(c_1)$ e $\mu(c_2)$:

- Nel caso di due criteri *ridondanti*, ossia che forniscono grossomodo una misura della stessa caratteristica, il peso associato all'insieme dei due criteri deve essere minore della somma dei loro pesi individuali:

$$\mu(\{c_1, c_2\}) < \mu(c_1) + \mu(c_2)$$

Nel nostro caso, ad esempio, i criteri che valutano la distanza robot-frontiera e la quantità di batteria necessaria per raggiungere il punto stabilito sono di questo tipo, infatti quest'ultimo valore viene calcolato tenendo conto proprio della distanza.

- Nel caso di due criteri *sinergici*, ossia molto diversi tra loro, il peso associato all'insieme dei due criteri deve essere maggiore della somma dei loro pesi individuali:

$$\mu(\{c_1, c_2\}) > \mu(c_1) + \mu(c_2)$$

Considerando questa proprietà, una posizione che soddisfa i due criteri in modo ragionevole è preferita ad una che li soddisfa in modo sbilanciato, dato che una situazione in cui entrambi abbiano un valore di utilità elevato è, peraltro, abbastanza improbabile. Tra i criteri che consideriamo, ad esempio, la stima dell'area e la misura della distanza robot-frontiera sono sinergici in quanto valutano due caratteristiche distinte. In base a quanto appena detto, tra le possibili frontiere ne verrà scelta una che soddisfa bene entrambi i criteri piuttosto di una in cui l'area ha un valore stimato molto alto ma che si trova ad una grande distanza rispetto al robot o viceversa.

- Nel caso di due criteri *indipendenti* tra loro il peso associato all'insieme dei due criteri deve essere uguale alla somma dei loro pesi individuali:

$$\mu(\{c_1, c_2\}) = \mu(c_1) + \mu(c_2)$$

La media pesata è un caso particolare di MCDM in cui tutti i criteri sono considerati indipendenti tra loro.

Lo stesso principio può essere applicato, nello stesso modo, anche ad insiemi che comprendono più di due criteri.
Per rendere più chiaro il concetto, di seguito mostriamo un esempio di applicazione di MCDM fatto considerando, per semplicità, solo tre dei quattro criteri citati precedentemente (stima dell'area, distanza robot-frontiera

e batteria necessaria per il raggiungimento dell'obiettivo) e supponendo che i valori di utilità già normalizzati e disposti in ordine crescente, relativi ad ogni criterio, per la posizione p in esame, siano i seguenti:

$$u_{\texttt{batteria}}(p) = u_{(1)}(p) = 0.3$$
$$u_{\texttt{distanza}}(p) = u_{(2)}(p) = 0.4$$
$$u_{\texttt{area}}(p) = u_{(3)}(p) = 0.7$$

Come prima cosa è necessario assegnare i pesi agli insiemi di criteri. Tenendo conto delle regole relative alla loro dipendenza, abbiamo scelto, a scopo dimostrativo, i pesi mostrati in Tabella 3.1[1].

CRITERI	PESI
$\emptyset$	0
area	0.6
distanza	0.3
batteria	0.1
area, distanza	0.95
area, batteria	0.7
distanza, batteria	0.2
area, distanza, batteria	1

Tabella 3.1: Esempio di possibili pesi associati ai criteri

La formula risultante per il calcolo dell'utilità della posizione p è mostrata di seguito:

$$\begin{aligned} u(p) = {} & (u_{(1)}(p) - u_{(0)}(p)) \cdot \mu(\texttt{area, distanza, batteria}) + \\ & + (u_{(2)}(p) - u_{(1)}(p)) \cdot \mu(\texttt{area, distanza}) + \\ & + (u_{(3)}(p) - u_{(2)}(p)) \cdot \mu(\texttt{area}) = \\ & = (0.3 - 0) \cdot 1 + (0.4 - 0.3) \cdot 0.95 + (0.7 - 0.4) \cdot 0.6 = 0.575 \end{aligned}$$

[1] Questi pesi non sono gli stessi che abbiamo usato nel nostro software, ma sono valori scelti a caso, seppur rispettando le regole di dipendenza e supponendo sinergici i criteri `area` e `distanza` e ridondanti `distanza` e `batteria`, per mostrare il procedimento del calcolo dell'utilità.

L'approccio MCDM è indipendente dalla natura e dal numero dei criteri che si considerano, è infatti possibile aggiungerne altri senza dover studiare una formula diversa che li contempli tutti, come nel caso di chi ha basato la propria strategia su funzioni *ad hoc*. La dimostrazione di quanto detto verrà illustrata nella Sezione 4.4.

Capitolo 4

Architettura del sistema

4.1 Descrizione generale del software

Tutto il nostro sistema è basato sul software, che è possibile scaricare gratuitamente dal sito ufficiale di Amsterdam Oxford Joint Rescue Forces[1] nella sezione downloads, sviluppato nel 2009 dalla squadra formata dai ricercatori delle università di Amsterdam e di Oxford, AOJRF, partecipante alla RoboCup Rescue Virtual Robot Competition.
Il programma, sviluppato usando il linguaggio di programmazione *Visual Basic .NET*, è molto ampio e include diverse strategie di esplorazione, alcune che prevedono l'intervento di un operatore umano e altre che lasciano i robot liberi di esplorare l'ambiente in modo autonomo. Le strategie sono gestite principalmente da due classi chiamate *Behavior*, che comprende tutti i possibili comportamenti eseguibili da ogni agente, e *Motion*, nei cui file viene effettivamente implementata la strategia. Ogni elemento della classe Behavior fa riferimento ad uno o più membri di Motion, attivandoli o disattivandoli a seconda della strategia che si vuole attuare. Ad ogni robot deve essere associato un comportamento iniziale, prima dell'avvio dell'esplorazione, che può poi essere modificato durante l'esecuzione scegliendo tra quelli disponibili. Tra questi alcuni esempi degni di nota sono *Follow Corridor Behavior* che, tramite l'attivazione del movimento *Corridor Walk*, fa in

[1] http://www.jointrescueforces.eu/

modo che l'agente si sposti muovendosi vicino ai muri, *Tele Operation*, che attivando *No Motion*, inibisce qualsiasi altra strategia utilizzata dal robot e permette all'operatore umano di comandarlo da remoto, prendendo decisioni al posto suo, e *Autonomous Exploration* la quale innesca *Frontier Exploration* come movimento principale, che si basa sulla formula:

$$u(f) = A(f) \cdot P(f)/d(f)$$

già discussa nella Sezione 2.3 e che abbiamo preso come punto di riferimento in questa tesi. Alcune strategie per gestire situazioni particolari riscontrate durante l'esplorazione sfruttano anche altri movimenti, ad esempio Autonomous Exploration utilizza anche *Avoid Team Mate* e *Avoid Victim* per evitare gli altri agenti e le vittime presenti nell'ambiente, nonché il già citato Corridor Walk come supporto tra un passo di esplorazione e l'altro.
Nello svolgimento del nostro lavoro abbiamo modificato solo una parte del codice originale, lasciando inalterato tutto ciò che riguarda gli algoritmi di costruzione della mappa, localizzazione del robot al suo interno, pianificazione del percorso, navigazione verso l'obiettivo scelto, ecc... I cambiamenti sono stati apportati solo alla parte relativa alla strategia di esplorazione Autonomous Exploration, ad eccezione di una piccola modifica, riguardante il punto da raggiungere una volta scelta una frontiera tra quelle candidate, fatta per risolvere un problema riscontrato durante l'esecuzione degli esperimenti e di cui parleremo nella Sezione 4.3. Praticamente abbiamo sostituito la nostra strategia, che abbiamo chiamato *POLIMI*, basata sull'approccio MCDM descritto nel Capitolo 3, ad Autonomous Exploration, lasciando inalterato tutto il resto.
Prima di descrivere il sistema nel dettaglio dobbiamo specificare che il software si compone di due eseguibili chiamati *UsarCommander* e *UsarClient* che permettono di sfruttarne le funzionalità in due modi diversi:

- *Controllo diretto*: non viene considerata la comunicazione tra gli agenti, i quali, quindi, non conoscono le posizioni degli altri componenti della squadra e non condividono con essi le informazioni riguardanti la mappa. Per l'utilizzo in questa modalità è sufficiente ese-

guire UsarCommander che visualizza la schermata principale del programma e permette di disporre, configurare e controllare tutti i robot nell'ambiente.

- *Controllo indiretto*: la comunicazione è l'elemento cruciale per un'efficiente esecuzione del compito di esplorazione. Comunicando, infatti, gli agenti possono condividere tutto ciò che riguarda la mappa e la propria posizione. In questo caso UsarCommander viene utilizzato solamente per la configurazione dei robot, e ad ogni componente della squadra a cui è affidato il compito di esplorazione è necessario associare uno UsarClient che permette di posizionare e gestire i robot all'interno dell'ambiente da linea di comando. Per la configurazione e il funzionamento tramite controllo indiretto si rimanda all'Appendice A.

Il controllo indiretto, che è, indubbiamente, il metodo più completo, nonché quello che viene adottato da AOJRF durante la RoboCup Rescue Virtual Robot Competition, è quello su cui ci siamo basati per lo svolgimento di tutti gli esperimenti. In questa modalità, la squadra sul campo deve essere composta da due o più agenti, uno dei quali è necessario che sia una ComStation, mostrata in Figura 4.1(a), che funge da collettore per l'invio e la ricezione di informazioni. Gli altri robot possono essere scelti tra tutti i modelli messi a disposizione da USARSim. Tra quelli disponibili noi, per l'esecuzione dei nostri esperimenti, abbiamo scelto il P2AT, la cui versione simulata si può vedere in Figura 4.1(b), visto che è proprio lo stesso tipo di robot utilizzato da AOJRF in alcune attività sperimentali [23].
La comunicazione viene simulata tramite *Wireless Simulation Server* (WSS) [15], un software sviluppato appositamente per questo scopo, dalla Jacobs University di Brema, per USARSim. I messaggi scambiati tra robot e ComStation sono di diversa natura. Alcuni servono ai robot per comunicare la propria posizione, eventuali collisioni contro gli ostacoli presenti nell'ambiente, la posizione delle vittime, le decisioni prese per quanto riguarda il punto da raggiungere e, in generale, tutto ciò che riguarda il loro stato. Altri vengono inviati dalla ComStation ai robot per aggiornarli sulla posizione e lo stato degli altri componenti della squadra. L'aspetto più importante da no-

tare è che i robot inviano alla ComStation anche le informazioni riguardanti la mappa, rilevate durante la navigazione. Spetta poi a quest'ultima il compito di unire tutte le mappe parziali in un'unica mappa globale che viene trasmessa a tutti gli agenti e visualizzata sullo schermo in tempo reale.

(a) ComStation (b) P2AT

Figura 4.1: Robot simulati

4.1.1 Illustrazione della procedura di esplorazione

Il funzionamento del sistema che, ovviamente, è uguale sia nel caso in cui venga usata POLIMI o Autonomous Exploration viene illustrato nella Figura 4.2 dove è indicata nel dettaglio solo la parte che riguarda la strategia di esplorazione. Lo schema a blocchi mostra la procedura eseguita da un singolo robot P2AT, che da ora in poi chiameremo Robotα, ma che è la stessa per tutti i componenti della squadra, durante l'esecuzione del compito di ricognizione.
Appena il Robotα viene posizionato all'interno della mappa si attiva la strategia di esplorazione, la quale prevede che il robot effettui diverse azioni in sequenza. Innanzitutto deve rilevare tutte le frontiere presenti nell'ambiente nell'istante di attuazione della strategia e valutare i criteri relativi all'area stimata, alla distanza, alla comunicazione e alla batteria, menzionati nel Capitolo 3, per ognuna di esse per quanto riguarda se stesso ma anche per quanto riguarda gli altri robot, se ce ne sono. Quindi, ad esempio, nel caso del criterio che misura la distanza tra il robot e la frontiera, il Robotα cal-

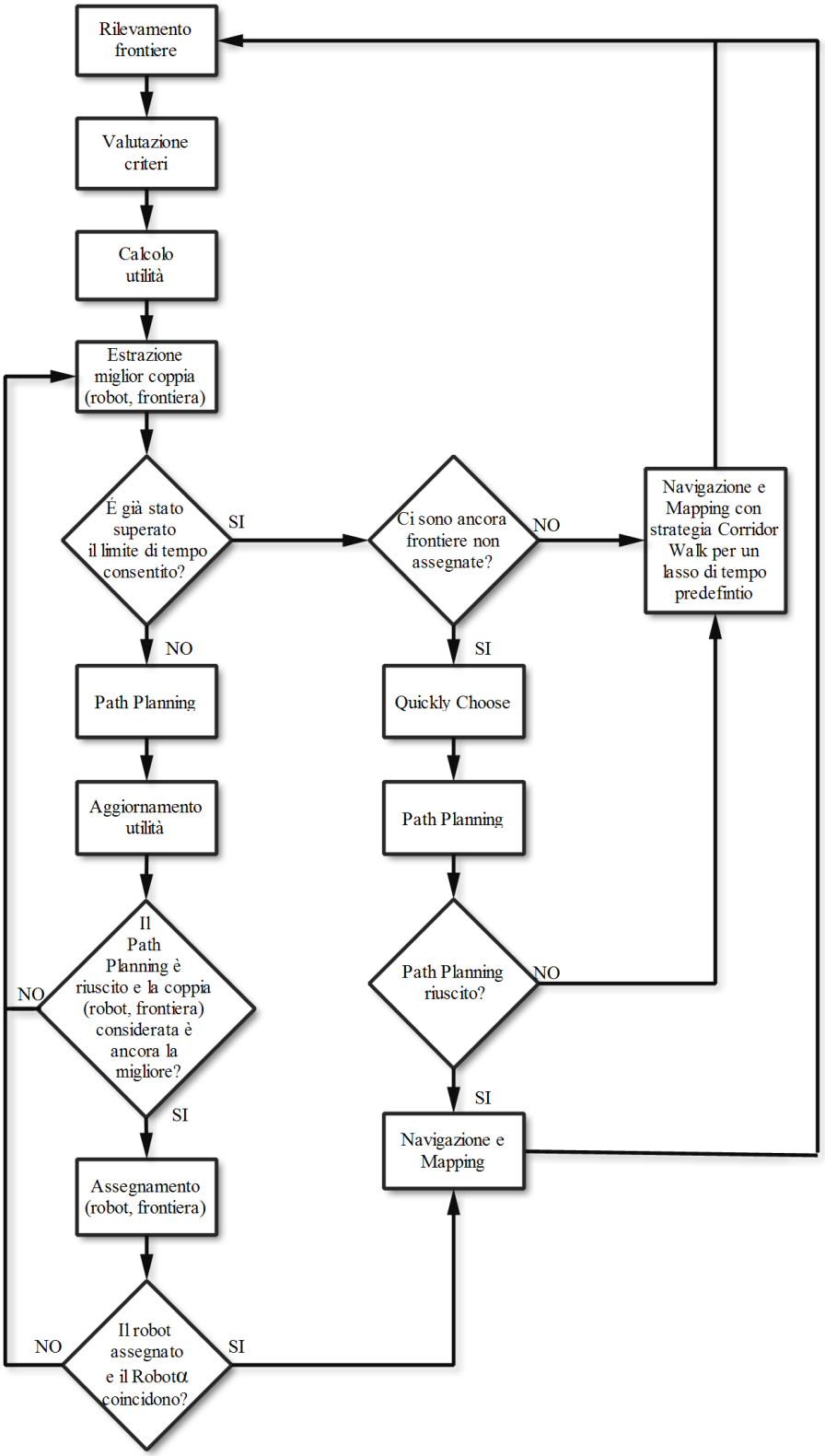


Figura 4.2: Schema a blocchi relativo al funzionamento del sistema per un singolo robot

cola sia la distanza tra sé e tutte le frontiere presenti che quella tra queste e tutti gli altri componenti della squadra. A questo punto dell'esecuzione la distanza considerata è quella euclidea, ossia la misura del segmento che unisce i due punti in linea retta, che fornisce un limite inferiore dell'effettiva distanza da percorrere. Con l'ausilio delle informazioni appena raccolte può procedere con il calcolo dell'utilità per ogni possibile coppia (robot, frontiera). Questi valori vengono posti all'interno di una matrice le cui righe e le cui colonne corrispondono rispettivamente ai robot e alle frontiere presenti. Dopo aver calcolato tutte le utilità il Robotα cerca, per ogni agente, quella maggiore individuando così la frontiera che rappresenta la prima scelta di tale agente. Se è presente più di una posizione candidata, per lo stesso agente, determina anche il secondo maggior valore di utilità rilevando la frontiera che rappresenta la sua seconda scelta. Fatto questo procede con l'estrazione della coppia (robot, frontiera) migliore tra tutte quelle presenti, vale a dire quella a cui è associato il valore di utilità più elevato in assoluto e attiva un algoritmo di Path Planning tramite il quale cerca di trovare il percorso più corto che permette al robot di raggiungere la frontiera evitando gli ostacoli presenti nella mappa. A questo punto il Robotα ricalcola il valore di utilità della coppia che sta considerando. Tale aggiornamento può avvenire in uno dei seguenti due modi:

- Se il Path Planning va a buon fine l'utilità viene ricalcolata utilizzando, al posto della distanza euclidea, la lunghezza del percorso ottenuto. Sicuramente la distanza calcolata con questo algoritmo è uguale o maggiore rispetto alla distanza euclidea considerata prima e, dato che la distanza incide negativamente sul calcolo dell'utilità, questo si traduce in un valore di utilità uguale o minore di quello precedente.

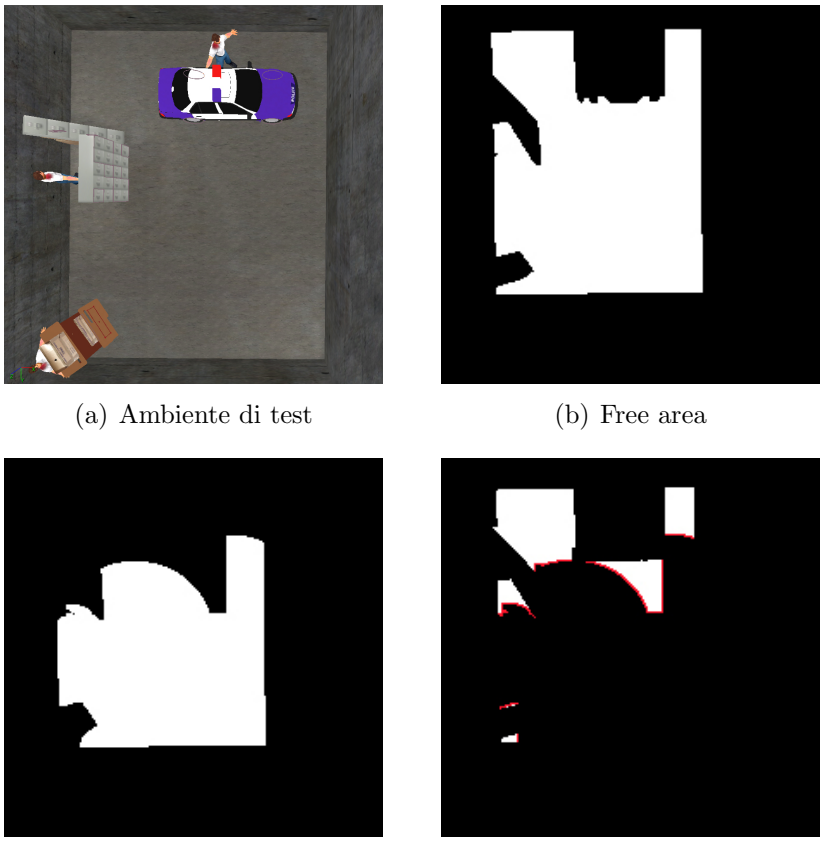
- In caso contrario, se la frontiera non è raggiungibile, l'utilità viene peggiorata per far si che questa coppia (robot, frontiera) non venga più scelta. Tale peggioramento viene fatto assegnandole un valore calcolato in base a quello della seconda miglior scelta associata allo stesso agente, se è presente, e comunque inferiore a questo. Altrimenti l'utilità viene posta uguale al valore di inizializzazione.

A questo punto, se il Path Planning è andato a buon fine e se alla coppia in esame è ancora associato il valore di utilità più elevato, il Robotα assegna la frontiera al robot e non li considera più fino al termine del passo di esecuzione corrente dell'algoritmo di esplorazione. In altre parole elimina dalla matrice delle utilità la riga e la colonna corrispondenti al robot e alla frontiera. Se, invece, la frontiera non è raggiungibile o, pur essendolo, l'utilità è peggiorata notevolmente e la coppia considerata non è più la migliore, oppure, nonostante l'assegnamento sia andato a buon fine, il robot associato alla frontiera non è il Robotα, vengono ripetuti in sequenza tutti i passi appena descritti partendo dall'estrazione della nuova miglior coppia (robot, frontiera). Quando il robot preso in considerazione ed associato alla frontiera è il Robotα l'elaborazione della strategia di esplorazione termina e vengono attivati gli algoritmi di navigazione verso l'obiettivo scelto e di costruzione della mappa. Dato che il Path Planning è un processo computazionalmente dispendioso, per rendere l'algoritmo di esplorazione più efficiente, come si può notare, non tutte le utilità vengono calcolate utilizzando la distanza reale ma solo quelle che assumono i valori più elevati considerando la distanza euclidea vengono aggiornate.
Un altro aspetto da sottolineare è dato dal fatto che il Robotα, durante l'esecuzione, è consapevole della presenza di altri robot nell'ambiente, tanto che calcola anche le utilità ad essi relative, per tutte le frontiere. Quando assegna un robot della squadra ad una frontiera non comunica ad altri questa informazione (quindi il robot non si recherà realmente verso l'obiettivo), e la utilizza per cercare di non assegnare a se stesso una frontiera che potrebbe essere preferibile associare ad un altro agente.
Per evitare che un robot si soffermi troppo sulla scelta del punto da raggiungere, però, quando viene superato il limite di tempo massimo consentito al robot per decidere, pari a venti secondi, se sono presenti frontiere che non sono ancora state associate a nessun componente della squadra, viene attivato un comportamento chiamato *Quickly Choose*. In questo caso il Robotα non considera più gli altri agenti ma solo se stesso e sceglie la frontiera, tra quelle non ancora assegnate, con il più alto valore di utilità. Successivamente, dopo aver stabilito l'obiettivo da raggiungere, effettua il Path Planning e, se que-

sto dà esito positivo, esegue le procedure di navigazione e mapping. Qualora allo scadere del limite di tempo, tutte le frontiere siano state assegnate o nel caso in cui, sebbene ci sia almeno una frontiera, il Path Planning abbia avuto esito negativo, si attiva una nuova strategia di esplorazione, chiamata Corridor Walk, che agisce in modo completamente diverso, infatti il robot avanza in linea retta finché non trova un ostacolo, dopodiché procede seguendone il bordo. Il Robotα naviga nell'ambiente e crea la mappa adottando questa strategia per un lasso di tempo preimpostato, di durata pari ad un minuto, quindi riattiva la strategia principale, la quale inizia con l'estrazione delle frontiere, che stavolta saranno diverse da quelle rilevate al passo precedente.

4.1.2 Definizione delle frontiere

Le frontiere a cui ci siamo riferiti fino ad ora sono, come già accennato nella Sezione 2.3, il confine tra una porzione dell'ambiente chiamata *safe area* o regione sicura e una detta *free area* o spazio libero. Sul robot, tra gli altri, sono montati due telemetri laser per cui la safe area può essere definita come la zona all'interno del campo d'azione di quello con range minimo, mentre la free area racchiude tutto ciò che viene rilevato dal sensore con range massimo. Ne deriva, quindi, che la regione sicura è un sottoinsieme dello spazio libero. Nella Figura 4.3 viene mostrato un esempio di estrazione delle frontiere fatto da un robot P2AT, durante l'esplorazione in un ambiente di test, di modeste dimensioni, utilizzato solo per questo scopo. Nella Figura 4.3(d), ottenuta tramite la sottrazione della Figura 4.3(c) dalla Figura 4.3(b), sono indicate, in rosso, le frontiere estratte in un dato istante e, in bianco, l'area presente al di là di esse. L'insieme dei possibili obiettivi raggiungibili nel passo di esplorazione corrente è composto dall'unione di tutte le nuove frontiere che si sono create come conseguenza della navigazione eseguita per raggiungere il punto attuale e di quelle presenti in precedenza ma che non sono state visitate da nessun robot. Inoltre, se viene utilizzato più di un agente, esclusa la ComStation, ogni componente della squadra condivide con gli altri l'insieme delle frontiere rilevate nella porzione di mappa che ha visitato, così che tutti considerano gli stessi punti candidati per scegliere il prossimo obiettivo.



(a) Ambiente di test

(b) Free area

(c) Safe area

(d) Frontiere estratte e rispettive aree

Figura 4.3: Processo di estrazione delle frontiere

Oltre ad aver sviluppato diverse versioni del nostro software, che verranno trattate nel dettaglio nelle sezioni seguenti unitamente alle motivazioni che hanno portato alla loro realizzazione, abbiamo implementato anche altre due strategie di esplorazione, anch'esse basate sul codice di AOJRF e descritte nella Sezione 4.5. La loro creazione si è resa necessaria per poter avere risultati relativi ad altre strategie, oltre a quella sviluppata da AOJRF, con cui confrontare quelli ottenuti dagli esperimenti effettuati utilizzando il nostro programma.

4.2 Prima versione di POLIMI

La prima versione del nostro software, presente nel file compresso `POLIMI V.1.zip` all'interno del DVD allegato, è stata sviluppata con lo scopo di implementare la strategia basata sull'approccio MCDM come metodo per valutare le posizioni candidate nell'ambito di un'esplorazione per una squadra di robot. Partendo dal codice scritto da AOJRF, che si può trovare nel file compresso `AOJRF2009 V.1.zip`, abbiamo apportato dei cambiamenti all'interno di tutti i file che riguardano la strategia Autonomous Exploration per poterla completamente sostituire con POLIMI e abbiamo modificato quelli che gestiscono l'interfaccia grafica del programma in modo da poter avere, una volta avviato, la possibilità di scegliere la nostra strategia tra quelle messe a disposizione.
La prima differenza tra il nostro programma e quello originale riguarda il numero di punti che abbiamo voluto considerare come possibili obiettivi e per i quali abbiamo calcolato l'utilità. La decisione presa è stata quella di analizzare tutte le frontiere presenti in ogni istante, al contrario di quanto è stato fatto da AOJRF. Nel codice originale, infatti, ad ogni passo di esplorazione innanzitutto viene determinata una soglia massima, che assume il valore dell'area maggiore tra quelle associate alle frontiere che fanno parte di un insieme che ne comprende solo il 36,9% (pari a $\frac{1}{e}$%). Successivamente, nel calcolo dell'utilità, vengono considerate, tra tutte le frontiere presenti nel passo di esplorazione corrente, solo quelle a cui è associata un'area maggiore o uguale rispetto a questa soglia.

4.2.1 Criteri utilizzati

Per la realizzazione di questa versione del codice abbiamo preso in considerazione gli stessi tre criteri utilizzati da AOJRF per valutare l'utilità delle frontiere presenti tra quelle candidate ad essere scelte come prossimo punto da raggiungere, dato che sono dei parametri significativi e adatti per essere sfruttati nello svolgimento di questo compito. Di seguito descriviamo nel dettaglio i tre criteri, già menzionati nel Capitolo 3:

- *Area*: valuta l'estensione, in metri quadrati, della porzione di mappa che si trova dietro alla frontiera che si sta considerando. Il processo di valutazione dell'area è strettamente legato a quello di estrazione delle frontiere descritto precedentemente, infatti, proprio nel fare questo il robot calcola ed associa ad ogni frontiera un valore che stima la quantità di area ad essa relativa, colorata di bianco nella Figura 4.3(d). Inoltre, per ognuna di esse, viene determinato il centro di tale area, ossia il punto che viene effettivamente raggiunto dal robot una volta scelto il suo obiettivo e che serve per la valutazione degli altri criteri.

- *Distanza*: misura, in metri, la distanza che divide il robot dalla frontiera in esame. La distanza viene calcolata non considerando un punto sulla frontiera, bensì il centro dell'area ad essa relativa. Questo criterio viene calcolato due volte, la prima per tutte le coppie (robot, frontiera), utilizzando la distanza euclidea, la seconda nel passo di aggiornamento, solo per le coppie a cui è associato un valore di utilità elevato, utilizzando la misura di distanza restituita dal Path Planning.

- *Comunicazione*: stima con quale probabilità il robot riuscirà a comunicare con la ComStation una volta raggiunto il punto al centro dell'area relativa alla frontiera oggetto di valutazione. Questo valore è calcolato, quindi, in funzione della distanza che separa la ComStation da tale posizione.

4.2.2 Scelta dei pesi

Il passo iniziale svolto per l'effettiva implementazione della nostra strategia è stata la scelta, fatta rispettando le regole elencate nel Capitolo 3, nonché l'introduzione nel codice, dei pesi, mostrati nella Tabella 4.1, per ogni possibile insieme di criteri.
Dato che l'obiettivo principale dell'esplorazione è quello di esaminare più area possibile all'interno dell'ambiente, abbiamo deciso di dare più importanza al criterio che valuta l'area visitabile una volta raggiunta la posizione scelta rispetto a quello che misura la distanza tra robot e frontiera. Alla comunica-

zione abbiamo assegnato un peso inferiore a quello associato a quest'ultimo in quanto, pur considerando il fatto che serve ai robot per scambiarsi informazioni, è stato ritenuto meno importante rispetto agli altri criteri per la valutazione dell'utilità e inoltre, nei nostri esperimenti, che sono fatti su mappe di dimensioni non eccessive, la comunicazione, anche se con diversi valori di probabilità, è quasi sempre garantita.

CRITERI	PESI
∅	0
area	0.5
distanza	0.3
comunicazione	0.2
area, distanza	0.95
area, comunicazione	0.7
distanza, comunicazione	0.4
area, distanza, comunicazione	1

Tabella 4.1: Pesi utilizzati nella prima versione di POLIMI

Per verificare di aver scelto dei pesi adeguati per il nostro scopo abbiamo effettuato alcuni esperimenti dando un peso maggiore alla distanza rispetto all'area e altri esperimenti modificando i pesi dei singoli criteri di una quantità pari al 10% del loro valore (e variando, in entrambi i casi, di conseguenza i pesi degli insiemi di criteri che ne comprendono più di uno). I risultati non sono mai stati migliori di quelli ottenuti con i pesi originali, soprattutto nelle prove in cui abbiamo dato più importanza alla distanza rispetto all'area visitabile.

Nonostante il fatto che ogni volta che si vuole modificare un peso è necessario ricompilare l'intero programma, durante lo svolgimento del lavoro abbiamo deciso, per semplicità, di introdurre i pesi direttamente nel codice e non in un file separato. Questo anche perché le operazioni di lettura da un documento esterno sono computazionalmente dispendiose e, siccome il software richiede già abbondanti risorse e durante un esperimento ogni robot dovrebbe accedervi un elevato numero di volte, abbiamo deciso di evitarle.

4.2.3 Implementazione della funzione MCDM

Una delle parti più importanti del nostro lavoro è stata l'inserimento nel programma della funzione relativa a MCDM per il calcolo delle utilità, introdotta nel Capitolo 3. Per poter gestire i criteri nel modo corretto li abbiamo inseriti in una struttura che ne associa il nome al valore:

```
Structure Criteria
    Dim name As String
    Dim value As Double
End Structure
```

Tutte le frontiere presenti e tutti gli agenti facenti parte della squadra, invece, sono contenuti in due strutture dati, chiamate rispettivamente `infos` e `_TeamMembers`, che vengono fatte scorrere per poter calcolare l'utilità per ogni coppia (robot, frontiera). Le linee di codice presentate di seguito mostrano come abbiamo deciso di implementare la formula principale:

```
For j As Integer = 0 To infos.Length - 1
    For i As Integer = 0 To Me.Control.Agent._TeamMembers.Count - 1
        ...
        ...
        area_norm = (info.Area - min_area) / (max_area - min_area)
        comm_success_norm = (comm_success(j) - min_comm_success) / (
            max_comm_success - min_comm_success)
        dist_robot_norm = 1 - ((dist_robot(i, j) - min_dist_robot) / (
            max_dist_robot - min_dist_robot))

        area.value = area_norm
        communication.value = comm_success_norm
        distance.value = dist_robot_norm

        Dim list_of_criteria() As Criteria = {area, communication, distance}
        Dim list_of_criteria_ordered(list_of_criteria.Length - 1) As
            Criteria
        Dim min_temp As Criteria
        Dim index As Integer

        index = 0
        min_temp = list_of_criteria(0)

        For h As Integer = 0 To list_of_criteria_ordered.Length - 1
            For k As Integer = 0 To list_of_criteria.Length - 1
                If list_of_criteria(k).value < min_temp.value Then
                    min_temp = list_of_criteria(k)
                    index = k
                End If
```

```
            Next k
            list_of_criteria_ordered(h) = min_temp
            list_of_criteria(index).value = 2
            min_temp = list_of_criteria(index)
        Next h

        util = (list_of_criteria_ordered(0).value - 0) * 1

        Dim temp As Double
        Dim weight_temp As Double

        temp = list_of_criteria_ordered(0).value
        For k As Integer = 1 To list_of_criteria_ordered.Length - 1
            weight_temp = SearchWeight(list_of_criteria_ordered , k)
            util = util + (list_of_criteria_ordered(k).value - temp) *
                weight_temp
            temp = list_of_criteria_ordered(k).value
        Next k

        utils(i, j) = util
        ...
        ...
    Next i
Next j
```

Per raggiungere il nostro scopo, per ogni coppia (robot, frontiera), abbiamo, prima di tutto, normalizzato i criteri ad essa associati estraendo il valore massimo e minimo, relativi ad ogni criterio, necessari per eseguire la procedura di normalizzazione. Ciò è stato fatto adottando un piccolo accorgimento, ossia sommando al valore massimo una quantità pari a 0.0001 nel caso fosse uguale a quello minimo, per evitare la divisione per zero. Fatto questo ci siamo serviti di due array per ordinare i criteri normalizzati e relativi alla coppia in esame, in base al loro valore, dal più piccolo al più grande. Innanzitutto li abbiamo inseriti in quello chiamato `list_of_criteria()` che viene fatto scorrere per individuare il criterio con valore minore. Quest'ultimo viene salvato nel primo posto libero del secondo array, `list_of_criteria_ordered()`, e il suo valore, nell'array da cui è stato estratto, viene impostato ad un numero abbastanza alto per fare in modo che non sia mai più scelto. Successivamente, ricordando che, come già detto nel Capitolo 3, l'ordinamento deve seguire la regola:

$$u_{(0)}(p) = 0 \leq u_{(1)}(p) \leq \ldots \leq u_{(n)}(p) \leq 1$$

siamo passati all'effettivo calcolo dell'utilità, la cui formula viene costruita per gradi. Il primo elemento si ottiene sottraendo $u_{(0)}(p)$ (che è zero per definizione) al valore del primo criterio nell'array `list_of_criteria_ordered()` e moltiplicando il tutto per il peso dell'insieme che contiene tutti i criteri (che è uno per definizione). Gli altri componenti della sommatoria, da inserire nel calcolo dell'utilità, si ottengono facendo scorrere l'array ordinato, partendo dal secondo elemento che lo compone, sottraendo al valore del criterio preso in considerazione nel passo corrente quello che lo precede nell'ordinamento e moltiplicando il risultato per il peso associato all'insieme di criteri corretto definito da:

$$A_{(j)} = \{i \in N | u_{(j)}(p) \leq u_{(i)}(p) \leq u_{(n)}(p)\}$$

Per recuperare il peso ad esso associato abbiamo sviluppato la funzione `SearchWeight()` mostrata di seguito:

```
Private Function SearchWeight(ByVal list_of_criteria_ordered() As Criteria,
    ByVal k As Integer) As Double
    Dim string_temp As String
    Dim bool As Boolean

    For j As Integer = 0 To weight.Count - 1
        bool = True
        string_temp = weight(j).name

        For i As Integer = k To list_of_criteria_ordered.Length - 1
            If Not string_temp.Contains(list_of_criteria_ordered(i).name)
                Then
                bool = False
            End If
        Next i

        If bool = True Then
            Return weight(j).value
        End If
    Next j

    Console.WriteLine("ERRORE nella ricerca del peso. Valore restituito: 0")

    Return 0
End Function
```

La variabile `weight` rappresenta la lista che contiene il peso associato agli insiemi di criteri. Ad ogni elemento di questa lista è stato collegato un iden-

tificativo che comprende i nomi dei criteri che compongono l'insieme e un valore che ne rappresenta il peso. Perché il codice funzioni correttamente il nome deve essere inserito nel formato "criterio1, criterio2, . . . , criterion" ed è necessario includere prima gli insiemi formati da un solo elemento, seguiti da quelli che ne contengono due e così via.
Tutto il procedimento appena descritto per la valutazione delle utilità viene eseguito per ogni possibile coppia (robot, frontiera) presente nel passo di esplorazione corrente, senza, ovviamente, considerare nel calcolo la ComStation.

4.2.4 Estrazione delle utilità migliori

Un'ulteriore variazione da noi apportata al programma originale, e che merita di essere menzionata, riguarda l'estrazione della prima e della seconda miglior frontiera per ogni agente e della miglior coppia (robot, frontiera) fatta cercando, nella matrice contenente tutte le utilità, quella maggiore. Per spiegare la differenza che intercorre tra la nostra implementazione e quella sviluppata da AOJRF riportiamo le linee di codice per l'esecuzione di questa operazione relative al nostro lavoro dove abbiamo indicato con dei commenti le parti in cui sono stati apportati cambiamenti o effettuate aggiunte e mostriamo con due esempi i risultati della loro applicazione.

```
For j As Integer = 0 To infos.Length - 1
    For i As Integer = 0 To Me.Control.Agent._TeamMembers.Count - 1
        ...
        ...
        If util >= bestmemberutil(i) Then    'sostituito >= a >

            goodmemberutil(i) = bestmemberutil(i)
            bestmemberutil(i) = util
            'nuovo assegnamento aggiunto da noi
            goodmemberfrontier(i) = bestmemberfrontier(i)
            bestmemberfrontier(i) = j
            If util >= bestutil Then     'sostituito >= a >
                bestutil = util
                bestmember = i
                bestfrontier = j
            End If

        End If
```

```
        'nuovo controllo aggiunto da noi
        If util > goodmemberutil(i) AndAlso util < bestmemberutil(i) Then
            goodmemberutil(i) = util
            goodmemberfrontier(i) = j
        End If
        ...
        ...
    Next i
Next j
```

Qui le quattro variabili `bestmemberutil(i)`, `bestmemberfrontier(i)`, `goodmemberutil(i)` e `goodmemberfrontier(i)` indicano rispettivamente il valore di utilità e l'indice delle frontiere che rappresentano la prima e la seconda scelta per il robot i-esimo. Inoltre, mentre `util` esprime il valore dell'utilità appena calcolata per una coppia (robot, frontiera), `bestutil`, `bestmember` e `bestfrontier` sono variabili relative al valore, al robot e alla frontiera associati alla miglior utilità in assoluto calcolata nel corrente passo di esplorazione. Nelle Tabelle 4.2 e 4.3 riportiamo dei valori di utilità fittizi relativi ad un robot e tre frontiere e nella Tabella 4.4 mostriamo i risultati ottenuti dopo l'applicazione dei due diversi metodi.
Si può notare che con l'applicazione del programma originale, in entrambi gli esempi, si ottengono gli stessi risultati, mentre col nostro software i valori estratti sono più conformi a quanto ci aspettiamo. Nel primo esempio, infatti, rileviamo come seconda miglior scelta la frontiera $F3$ con utilità pari a 13 che, invece, non viene nemmeno considerata col software di AOJRF, e, nel secondo esempio, sostituendo semplicemente un $>$ con un $\geq$, facciamo in modo che se si hanno due frontiere con lo stesso valore di utilità ne venga selezionata una ($F3$) come la prima e l'altra ($F2$) come la seconda miglior scelta.

	FRONTIERA 1	FRONTIERA 2	FRONTIERA 3
ROBOT 1	10	15	13

Tabella 4.2: Valori di utilità fittizi per il primo esempio di confronto dei codici

	FRONTIERA F1	FRONTIERA F2	FRONTIERA F3
ROBOT R1	10	15	15

Tabella 4.3: Valori di utilità fittizi per il secondo esempio di confronto dei codici

	AOJRF	POLIMI ESEMPIO 1	POLIMI ESEMPIO 2
`goodmemberutil`	10	13	15
`bestmemberutil`	15	15	15
`bestmemberfrontier`	F2	F2	F3
`goodmemberfrontier`	–	F3	F2
`bestutil`	15	15	15
`bestfrontier`	F2	F2	F3

Tabella 4.4: Risultati ottenuti applicando il nostro codice e quello di AOJRF

4.2.5 Normalizzazione della distanza reale

Terminata la prima fase di calcolo dell'utilità, si procede con la determinazione, fatta tramite l'algoritmo di Path Planning, del percorso più breve che separa i due componenti della coppia (robot, frontiera) a cui è associata l'utilità maggiore. Se questa operazione termina con successo, il risultato ci fornisce l'indicazione esatta del valore associato al criterio che determina la distanza tra robot e frontiera (che nel passo precedente era stato approssimato utilizzando la distanza euclidea) e ci permette di utilizzarlo per effettuare l'aggiornamento dell'utilità. Il nostro procedimento per il calcolo di quest'ultima prevede che il criterio venga normalizzato e per fare ciò dovremmo essere in possesso di tutti i veri valori di distanza che separano ogni robot da ogni frontiera. Se nella fase appena conclusa non è stato un problema calcolare tutti i valori di distanza, in questa lo è in quanto eseguire il Path Planning per ogni coppia (robot, frontiera) è proibitivo per il fatto che richiede troppe risorse. Abbiamo risolto questo problema utilizzando, per la normalizzazione del nuovo valore di distanza, i vecchi valori, massimo e minimo, usati precedentemente.

È possibile comprendere meglio tutto ciò attraverso un semplice esempio in cui si considera la presenza di cinque frontiere e un robot. La Tabella 4.5 mostra i valori di distanza euclidea calcolati rispetto ad ogni frontiera. Supponiamo che il valore di utilità globale più elevato sia associato alla frontiera numero 3 e che il nuovo valore di distanza ad essa relativo, calcolato col Path Planning, sia pari a $50m$. In questo caso la formula usata per la normalizzazione e quindi per calcolare l'utilità del criterio distanza per questa posizione sarebbe $u_d(3) = 1-((50-10)/(70-10))$. Come si può notare il procedimento considera ancora il valore minimo e massimo relativi alla distanza euclidea.

FRONTIERE	DISTANZA EUCLIDEA
1	10
2	15
3	**23**
4	30
5	70

Tabella 4.5: Valori di distanza per l'esempio di normalizzazione

Mentre non possono verificarsi casi in cui la distanza reale sia minore di 10, può invece succedere che tale valore risulti maggiore di 70. In questa situazione, visto che l'utilità associata al valore massimo è pari a zero, abbiamo deciso di porre anche $u_d(3) = 0$, ma bisogna sottolineare che questa nuova misura di distanza non sostituisce quella massima utilizzata fin'ora la quale resterà tale fino alla fine del corrente passo di esplorazione.
Dopo aver normalizzato il nuovo valore associato al criterio che misura la distanza del robot dalla frontiera si procede con l'aggiornamento dell'utilità seguendo lo stesso procedimento descritto precedentemente, ma utilizzando nel calcolo questo nuovo valore.

Dopo aver eseguito tutti i cambiamenti appena descritti e apportato altre piccole modifiche secondarie ma necessarie per far funzionare la nostra strategia

abbiamo effettuato una serie di esperimenti i cui risultati verranno illustrati e discussi nella Sezione 5.1.

4.3 Seconda versione di POLIMI

La creazione di una seconda versione del nostro programma, anch'essa presente nel DVD allegato e alla quale si può accedere estraendo il file `POLIMI V.2.zip`, si è resa necessaria come conseguenza dell'esecuzione degli esperimenti effettuati con quella precedente all'interno di una mappa che simula uno spazio aperto. In un ambiente simile, infatti, è possibile giungere in una situazione come quella mostrata in Figura 4.4 in cui il robot rimane bloccato nel punto scelto come obiettivo.

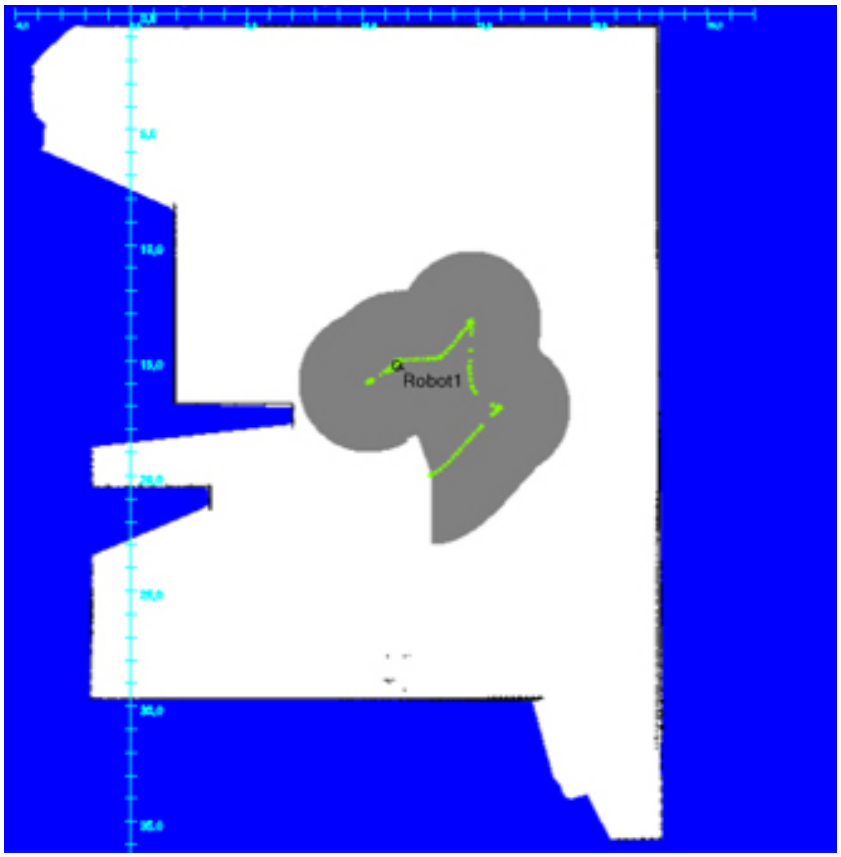


Figura 4.4: Problema in uno spazio aperto

Questo è dovuto proprio alla decisione di eleggere il centro dell'area relativa alla frontiera come posizione rappresentativa della frontiera stessa e a come questa viene determinata non verificando se il punto faccia o no parte dell'area safe (in grigio nell'immagine). Solitamente il robot muovendosi nell'ambiente rileva l'area free e l'area safe, che aumentano ogni volta che raggiunge nuovi obiettivi. Modificandosi, queste aree, definiscono nuove frontiere, tra le quali il robot può scegliere la migliore nel prossimo passo di esplorazione. In questo caso, invece, dopo aver visitato una piccola porzione della mappa, mentre si reca verso l'obiettivo selezionato nel passo corrente, il telemetro laser non rileva ulteriore spazio libero e l'area free non cambia. Per come è stato sviluppato l'algoritmo, questa situazione porta il robot a scegliere come prossimo punto da raggiungere lo stesso in cui si trova, bloccandolo per il resto dell'esecuzione. Per far proseguire l'esplorazione durante gli esperimenti siamo dovuti intervenire modificando la strategia da POLIMI a Tele Operation, così da poter comandare manualmente il robot, creando nuove frontiere e facendo in modo che non scegliesse più la stessa aumentando il numero di posizioni candidate, e, fatto questo, reimpostando POLIMI come strategia principale.
Nell'esempio riportato sopra sono presenti una sola frontiera e un solo agente ma il problema si può verificare anche in presenza di più componenti della squadra e di più punti candidati quando, nel raggiungerne uno, non cambiano le informazioni relative all'area free e questo resta comunque la migliore posizione per il robot.
Oltre che nello spazio aperto, come sarà spiegato nel Capitolo 5, abbiamo effettuato esperimenti anche utilizzando un'altra mappa, con caratteristiche molto diverse, raffigurante un piano di un hotel. Con la prima versione di POLIMI, il problema si è verificato solo nello spazio aperto, mentre non è successo niente del genere nell'altro ambiente. In un contesto come quello dell'hotel, infatti, è meno probabile incorrere in questo tipo di situazione in quanto durante l'esplorazione le frontiere si modificano maggiormente da un passo di esplorazione all'altro.
La porzione di codice relativa alla scelta del punto centrale della frontiera, sviluppata da AOJRF e utilizzata all'interno della prima versione del nostro

software, che causa il problema appena descritto è la seguente:

```
Public Function ExtractFrontierInfo(ByVal regions As Bitmap) As FrontierInfo
    ()
    Dim infos As New List(Of FrontierInfo)

    If regions.Width > 1 AndAlso regions.Height > 1 Then

        Dim counter As New BlobCounter(regions)
        Dim blobs() As Blob = counter.GetObjects(regions)

        Dim xCenter As Double, yCenter As Double, count As Integer
        For Each blob As Blob In blobs
            'compute true area in number of pixels
            count = 0
            xCenter = 0
            yCenter = 0

            Dim bbits As BitmapData = blob.Image.LockBits(New Rectangle(0,
                0, blob.Image.Width, blob.Image.Height), ImageLockMode.
                ReadOnly, PixelFormat.Format8bppIndexed)
            For j As Integer = 0 To bbits.Height - 1
                For i As Integer = 0 To bbits.Width - 1
                    If Marshal.ReadByte(bbits.Scan0, j * bbits.Stride + i) >
                        0 Then
                        count += 1
                        xCenter += i
                        yCenter += j
                    End If
                Next
            Next
            blob.Image.UnlockBits(bbits)

            'compute averages
            xCenter = xCenter / count + blob.Location.X
            yCenter = yCenter / count + blob.Location.Y

            infos.Add(New FrontierInfo(count / 100, xCenter, yCenter))
        Next

        'cleanup
        For Each blob As Blob In blobs
            blob.Dispose()
        Next
        blobs = Nothing

    End If

    Return infos.ToArray
End Function
```

La soluzione da noi impiegata, per fare in modo che il robot non rimanga bloccato durante l'esecuzione, consiste nel cambiare la posizione raggiungibile all'interno dell'area presente al di là della frontiera scelta. A questo proposito abbiamo deciso di sostituire il centro dell'area con un punto scelto a caso al suo interno, controllando anche che tale punto non faccia parte dell'area safe ma solo di quella free.
Il codice, con cui abbiamo sostituito quello originale e in cui, ad indicare il punto random selezionato, abbiamo mantenuto le variabili `xCenter` e `yCenter` già presenti, anche se non rappresentano più un punto centrale, è riportato di seguito:

```
Public Function ExtractFrontierInfo(ByVal regions As Bitmap) As FrontierInfo
    ()
    Dim infos As New List(Of FrontierInfo)

    If regions.Width > 1 AndAlso regions.Height > 1 Then

        Dim counter As New BlobCounter(regions)
        Dim blobs() As Blob = counter.GetObjects(regions)

        Dim xCenter As Double, yCenter As Double, count As Integer

        For Each blob As Blob In blobs
            'compute true area in number of pixels
            count = 0
            xCenter = 0
            yCenter = 0

            Dim bbits As BitmapData = blob.Image.LockBits(New Rectangle(0,
                0, blob.Image.Width, blob.Image.Height), ImageLockMode.
                ReadOnly, PixelFormat.Format8bppIndexed)

            For j As Integer = 0 To bbits.Height - 1
                For i As Integer = 0 To bbits.Width - 1
                    If Marshal.ReadByte(bbits.Scan0, j * bbits.Stride + i) >
                        0 Then
                        count += 1
                    End If
                Next
            Next
            blob.Image.UnlockBits(bbits)

            Dim xCenter_temp As Integer, yCenter_temp As Integer
            Dim found As Boolean
            found = False
```

```
            Dim rand As New Random

            While (found = False)
                xCenter_temp = rand.Next(blob.Image.Width + 1)
                yCenter_temp = rand.Next(blob.Image.Height + 1)

                'controlliamo che il punto random non faccia parte dell'area
                     safe
                If ((Not blob.Image.GetPixel(xCenter_temp, yCenter_temp).A
                    <> 255) AndAlso (Not blob.Image.GetPixel(xCenter_temp,
                    yCenter_temp).B <> 255) AndAlso (Not blob.Image.GetPixel
                    (xCenter_temp, yCenter_temp).R <> 255) AndAlso (Not blob
                    .Image.GetPixel(xCenter_temp, yCenter_temp).G <> 255))
                    Then
                    found = True
                End If
            End While

            xCenter = xCenter_temp + blob.Location.X
            yCenter = yCenter_temp + blob.Location.Y

            infos.Add(New FrontierInfo(count / 100, xCenter, yCenter))
        Next

        'cleanup
        For Each blob As Blob In blobs
            blob.Dispose()
        Next
        blobs = Nothing

    End If

    Return infos.ToArray
End Function
```

Facendo ciò non ci aspettiamo di migliorare le prestazioni in termini di area mappata, ma di raggiungere risultati simili a quelli già ottenuti e di non dover più intervenire manualmente sui movimenti dell'agente.
Abbiamo applicato la stessa modifica sia all'interno del nostro software che in quello sviluppato da AOJRF, che è possibile trovare nel file compresso `AOJRF2009 V.2.zip` del DVD allegato. Gli esperimenti effettuati utilizzando questa versione sono illustrati nella Sezione 5.2 in cui confrontiamo anche il numero di interventi manuali per sbloccare il robot che si sono resi necessari in questo caso con quelli effettuati usando la versione precedente.

4.4 Terza versione di POLIMI

La terza e ultima versione del codice da noi sviluppato serve per mostrare un grande vantaggio derivante dall'utilizzo di una strategia di esplorazione basata sull'approccio MCDM, ovvero la possibilità di introdurre un nuovo criterio, da prendere in considerazione per valutare l'utilità di una possibile posizione raggiungibile, senza dover studiare una nuova formula che lo comprenda e senza dover stravolgere il codice apportando troppe modifiche.
In questa fase abbiamo introdotto l'utilizzo della batteria e abbiamo fatto in modo che il robot, nel prendere le decisioni riguardo il prossimo obiettivo da raggiungere, contemplasse anche questo aspetto.
Simulare in modo realistico il funzionamento di una batteria non è banale e, dato che non abbiamo individuato buoni modelli per stimarne il consumo e che si tratta di un argomento che esula dall'obiettivo principale di questa tesi, abbiamo implementato le due seguenti soluzioni, le quali, pur non essendo particolarmente complesse, sono comunque adeguate per il raggiungimento del nostro scopo e per la dimostrazione della flessibilità di MCDM:

- La prima soluzione consiste nell'introduzione di un nuovo criterio che stima la percentuale di batteria necessaria al robot per riuscire a raggiungere una determinata frontiera.

- La seconda soluzione non riguarda l'utilizzo di un vero e proprio criterio, ma consiste nel considerare la presenza della batteria agendo sui pesi degli altri criteri presenti.

Con MCDM, quindi, abbiamo la possibilità di sfruttare due diverse modalità per prendere in esame una nuova informazione che influenza le scelte degli agenti all'interno dell'ambiente.
Di entrambi i metodi, spiegati nel dettaglio successivamente e i cui risultati verranno confrontati e discussi nella Sezione 5.3, abbiamo sviluppato due versioni che, come quelle precedenti, differiscono tra loro solo per il punto da raggiungere all'interno dell'area associata alla frontiera scelta, che può essere quello centrale o uno scelto casualmente. Nel DVD allegato è possibile trovare, estraendo i file `POLIMI V.3 batteria.zip` e `POLIMI V.3 batteria`

`random.zip`, il codice sviluppato per implementare la prima soluzione e, all'interno dei file `POLIMI V.3 doppi pesi.zip` e `POLIMI V.3 doppi pesi random.zip`, quello relativo alla seconda.

4.4.1 Batteria: introduzione di un nuovo criterio

Il primo metodo che abbiamo studiato per poter considerare, nella valutazione di una posizione candidata, anche la presenza della batteria consiste nell'introduzione di un nuovo criterio che si aggiunge ai tre già descritti precedentemente. Quest'ultimo serve per valutare la percentuale di scarica della batteria relativa ad un robot e alla frontiera che si stanno considerando ed è una misura che dipende strettamente dalla loro distanza. Il criterio prevede che si preferiscano posizioni per raggiungere le quali il robot debba consumare meno batteria possibile. Per simulare il funzionamento della batteria abbiamo deciso di utilizzare una semplice funzione, mostrata in Figura 4.5, in cui il livello $y \in [0, 1]$ della batteria decresce linearmente col passare del tempo x:

$$y = -\frac{1}{1200}x + 1$$

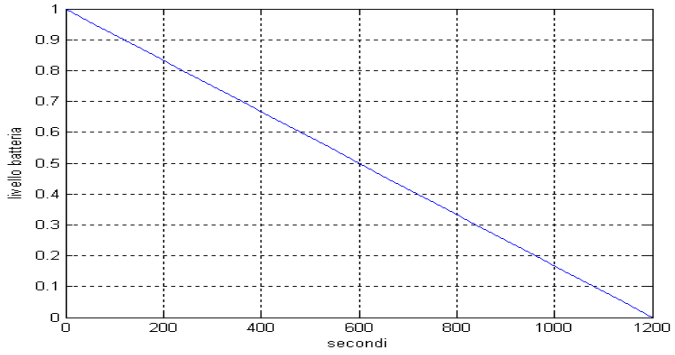


Figura 4.5: Andamento della funzione utilizzata per simulare la batteria

Poiché il valore di nostro interesse riguarda la percentuale di scarica della batteria, la procedura che ci fornisce questa misura, calcolata in base al tempo, è la seguente:

```
Private Function DischargeBattery(ByVal time As Double) As Double
    Dim discharge As Double
    discharge = 1 - (((-1 / 1200) * time) + 1)
    Return discharge
End Function
```

Il tempo considerato in questa operazione può assumere valori compresi tra 0 e 1200 secondi, che corrispondono alla durata massima di un'esecuzione del programma. È stato scelto questo lasso di tempo in quanto è quello utilizzato anche durante le gare della RoboCup Rescue Virtual Robot Competition. Come si può notare dalla definizione della funzione appena introdotta, la batteria non si scaricherà mai prima della fine dell'esplorazione e il criterio ad essa associato fornisce, quindi, informazioni equivalenti a quelle relative al tempo necessario ad un robot per raggiungere una determinata frontiera. Questa modellizzazione, come già accennato, è molto semplice ma ci permette ugualmente di conseguire risultati interessanti.
Il tempo di percorrenza per raggiungere una frontiera viene determinato in base alla distanza del robot da essa e alla velocità usata per raggiungerla. Dato che il criterio relativo alla distanza viene valutato due volte e in due modi diversi durante l'esecuzione, anche il tempo, che dipende da questa, e di conseguenza il criterio che misura il consumo della batteria, assumono due diversi valori nel corso dello svolgimento dell'algoritmo.
Nella prima fase di calcolo dell'utilità, in cui si considera la distanza euclidea, il tempo viene calcolato tramite la formula:

$$Tempo = \frac{DistanzaEuclidea}{VelocitàLineare}$$

Nella fase di aggiornamento dell'utilità, invece, conoscendo il percorso reale che permette al robot di raggiungere la frontiera, rilevato con l'algoritmo di Path Planning, abbiamo considerato nel calcolo del tempo il nuovo valore di distanza, nonché il tempo di rotazione, necessario al robot per effettuare tutte le svolte di cui il percorso è composto. Questo influisce parecchio sul calcolo del tempo complessivo in quanto, in presenza di una curva nel percorso, l'agente si ferma, esegue la rotazione e successivamente procede in rettilineo.

La formula utilizzata in questo frangente è la seguente:

$$Tempo = \frac{DistanzaPathPlanning}{VelocitàLineare} + TempoRotazione$$

Nei nostri esperimenti abbiamo sempre utilizzato robot di tipo P2AT a cui abbiamo applicato una velocità lineare di $2\frac{rad}{s}$. Per ottenere una misura del tempo in secondi, dovendo moltiplicare la velocità ad una distanza calcolata in metri, siamo stati costretti ad adottare alcuni accorgimenti per adattare le varie unità di misura. Nel caso della velocità lineare, sapendo che un radiante al secondo equivale a 9.5493 giri al minuto e che, dal manuale di USARSim [24], il diametro di una ruota del P2AT corrisponde a $0.22m$, abbiamo calcolato il suo valore effettivo, da utilizzare nel nostro programma, eseguendo pochi semplici calcoli:

$$CrfRuotaP2AT = DiametroRuotaP2AT \cdot \pi = 0.22 \cdot \pi = 0.6912\frac{m}{giri}$$
$$2\frac{rad}{s} = 2 \cdot 9.5493 = 19.0986\frac{giri}{min} = 0.3183\frac{giri}{s}$$
$$VelocitàLineare = 0.3183 \cdot 0.6912 = 0.22\frac{m}{s}$$

Il tempo di rotazione, invece, è una grandezza che dipende dall'angolo di rotazione complessivo, necessario affinché il robot possa affrontare tutti i cambiamenti di direzione presenti nel percorso che lo separa dalla frontiera. Per determinarlo abbiamo, innanzitutto, cronometrato il tempo impiegato da un P2AT per effettuare dieci giri completi su se stesso, equivalenti a $(10 \cdot 2 \cdot \pi)rad$, che è risultato pari a $290s$. Successivamente, sfruttando questa misura, abbiamo calcolato il tempo necessario al robot per ruotare di un singolo radiante[2]:

$$TempoRotazioneAlRadiante = \frac{290}{10 \cdot 2 \cdot \pi} = 4.6155\frac{s}{rad}$$

[2]Abbiamo deciso di optare per una misura in radianti, piuttosto che in gradi, in quanto la funzione utilizzata nel codice restituisce il valore dell'angolo espresso in questa unità di misura.

Questo valore è quello che viene moltiplicato all'angolo di rotazione per ottenere la misura che stiamo cercando. Abbiamo determinato l'angolo complessivo sfruttando l'insieme degli elementi restituiti dal Path Planning ognuno dei quali rappresenta una coppia di coordinate x e y intere, che definisce un punto in un piano bidimensionale. La procedura che abbiamo utilizzato consiste nell'estrarre ogni gruppo di tre punti contigui (A, B, C), individuare la rotazione necessaria al robot per andare da A a C passando per B, mostrata in Figura 4.6, e sommare tutti i valori trovati al fine di ottenere l'angolo totale.

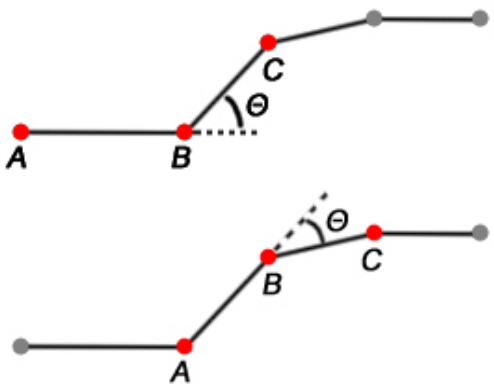


Figura 4.6: Spiegazione grafica della valutazione dell'angolo di rotazione

L'angolo θ corrispondente ad una singola rotazione viene calcolato tramite la formula:

$$\theta = \arccos \frac{\overline{AB} \cdot \overline{BC}}{\mid AB \mid \cdot \mid BC \mid}$$

All'interno del codice la funzione implementata per svolgere queste operazioni è la seguente:

```
Private Function RotationTime(ByVal path As Point()) As Double
    Dim x1, y1, x2, y2 As Double
    Dim cos_angle, angle, rotation, rotation_time As Double
    rotation = 0
    For i As Integer = 0 To path.Length - 3
        x1 = path(i + 1).X - path(i).X
        y1 = path(i + 1).Y - path(i).Y
```

```
        x2 = path(i + 2).X - path(i + 1).X
        y2 = path(i + 2).Y - path(i + 1).Y
        cos_angle = ((x1 * x2) + (y1 * y2)) / (Sqrt(Pow(x1, 2) + Pow(y1, 2))
            * Sqrt(Pow(x2, 2) + Pow(y2, 2)))
        angle = Acos(cos_angle)
        rotation = rotation + angle
    Next
    rotation_time = rotation * 4.6155
    Return rotation_time
End Function
```

Bisogna notare che questa versione del programma non può essere immediatamente utilizzata con un robot diverso dal P2AT in quanto, avendo caratteristiche diverse, si dovrebbero ricalcolare tutti i valori appena descritti.
L'utilizzo del criterio che ci fornisce la percentuale di scarica della batteria è molto simile a quello del criterio che misura la distanza, e, come per questo, è necessario determinarne il valore per ogni coppia (robot, frontiera). Per ognuna di queste, utilizzando il tempo necessario al robot per raggiungere una frontiera, abbiamo calcolato il consumo della batteria. Tra tutti i valori ottenuti abbiamo, poi, recuperato il minimo e il massimo per poter procedere con la normalizzazione fatta, sia per il primo calcolo dell'utilità che per il suo aggiornamento, utilizzando lo stesso procedimento svolto per normalizzare la distanza. Fatto questo, l'unica modifica apportata alla procedura per il calcolo dell'utilità è stata l'inserimento del nuovo criterio normalizzato all'interno dell'array contenente tutti i criteri presi in considerazione, senza dover apportare ulteriori modifiche o studiare una nuova formula comprendente anche la batteria:

```
...
Dim list_of_criteria() As Criteria={area, communication, distance, battery}
...
```

Con l'inserimento della batteria nella valutazione dei punti candidati abbiamo dovuto scegliere dei nuovi pesi, mostrati nella Tabella 4.6, da assegnare agli insiemi di criteri.
Anche in questo caso abbiamo deciso di dare più importanza all'area stimata e meno alla comunicazione. Come prima, alla distanza abbiamo associato un peso di valore intermedio, che, inoltre, coincide con quello del peso relativo al nuovo criterio. Abbiamo preso questa decisione in quanto entrambi, data la

loro forte dipendenza, ci forniscono delle informazioni simili e, quindi, della stessa importanza. Questa relazione di ridondanza è dimostrata anche dal peso associato all'insieme formato da questi due criteri che è minore della somma di quelli associati a distanza e batteria presi singolarmente.

CRITERI	PESI
$\emptyset$	0
area	0.4
distanza	0.25
comunicazione	0.1
batteria	0.25
area, distanza	0.75
area, comunicazione	0.5
area, batteria	0.65
distanza, comunicazione	0.25
distanza, batteria	0.35
comunicazione, batteria	0.25
area, distanza, comunicazione	0.75
area, distanza, batteria	0.9
area, comunicazione, batteria	0.75
distanza, comunicazione, batteria	0.45
area, distanza, comunicazione, batteria	1

Tabella 4.6: Pesi utilizzati con l'aggiunta del criterio batteria

4.4.2 Batteria: utilizzo di due diversi insiemi di pesi

Non considerando il criterio relativo alla batteria, a parità di mappa, posizione dei robot e quantità di area esplorata, la valutazione di un punto candidato non cambia col variare dell'istante di tempo in cui viene effettuata. Ciò, al contrario, non si verifica prendendo in considerazione anche il consumo di batteria nella determinazione della miglior frontiera. In questo caso, infatti, è necessario esaminare la storia passata dei robot e, quindi, conoscere come e quando ogni agente è giunto nella posizione in cui si trova e dalla quale deve scegliere il prossimo punto da raggiungere.

La seconda soluzione che abbiamo implementato per considerare la batteria consiste nel simulare il comportamento appena descritto utilizzando due istanze della nostra strategia, ognuna con un diverso insieme di pesi associato ai criteri considerati. In questo caso, i criteri sono gli stessi che abbiamo utilizzato per il calcolo dell'utilità nella prima versione del software, mentre, per quanto riguarda la batteria, invece di inserire un criterio vero e proprio che la rappresenti, abbiamo fatto in modo che, durante una normale esecuzione del programma, della durata di venti minuti, il robot assumesse due comportamenti differenti:

- Un comportamento più aggressivo durante i primi dieci minuti dell'esecuzione, nel corso dei quali consideriamo che la batteria abbia un livello di carica elevato e, sotto questa ipotesi, che il robot possa permettersi di visitare aree molto distanti rispetto al luogo in cui si trova.

- Un comportamento più conservativo durante i restanti dieci minuti dell'esecuzione, nel corso dei quali consideriamo che la batteria sia in fase di scarica e che, per questo motivo, il robot rimanga ad esplorare zone vicine alla sua posizione attuale.

Al fine di implementare queste due situazioni abbiamo adottato, come già detto, due diversi insiemi di pesi, mostrati nella Tabella 4.7.

CRITERI	PESI PRIMI 10 MINUTI	PESI RESTANTI 10 MINUTI
$\emptyset$	0	0
area	0.6	0.4
distanza	0.1	0.5
comunicazione	0.3	0.1
area, distanza	0.8	0.95
area, comunicazione	0.9	0.5
distanza, comunicazione	0.3	0.5
area, distanza, comunicazione	1	1

Tabella 4.7: Insiemi di pesi utilizzati per simulare la batteria

Per decidere quale dei due utilizzare nella valutazione dell'utilità ad ogni passo di esplorazione, abbiamo fatto in modo di controllare il tempo già trascorso dall'inizio dell'esecuzione, rilevato sfruttando un sensore di stato presente sui robot, e, in base a questa informazione, di selezionare l'insieme di pesi appropriato. Per usufruire correttamente di questo sensore è stato necessario inserire, nel file `USARBot.ini`[3] relativo alle impostazioni del robot P2AT, l'indicazione riguardo la durata massima della batteria, ossia `BatteryLife=1200`.
Scegliendo i pesi abbiamo fatto in modo di distinguere notevolmente i due comportamenti definendo per il criterio che valuta la distanza due pesi alquanto diversi tra loro. Come si può notare, infatti, nel primo caso abbiamo deciso di stabilire, per la distanza, un peso inferiore anche a quello relativo alla comunicazione, per evidenziarne la scarsa importanza e fare in modo che il robot seguisse percorsi più lunghi. Nel secondo caso, invece, per la distanza abbiamo impostato un peso che supera quello associato all'area stimata, affinché il robot potesse eseguire il comportamento contrario.
Un metodo come quello appena esposto, che consiste nel determinare due o più insiemi di pesi da associare ai criteri per sfruttare, oltre a questi, ulteriori informazioni nella valutazione dell'utilità, può essere utilizzato solo nel caso in cui questi nuovi elementi dipendano dalla storia passata della situazione in esame. Ad esempio, un'informazione come quella data dalla distanza che separa un robot da una frontiera non può, quindi, essere gestita in questo modo, ma solo adottando un criterio che la possa identificare. Quest'ultima, infatti, a parità di numero e dimensione delle frontiere, spazio esplorato e posizione dei robot, è una grandezza che non dipende da come l'agente è giunto nella situazione in cui si trova. Considerando la batteria, al contrario, tale dipendenza sussiste in quanto se il robot è in una certa posizione con la batteria abbastanza carica allora si può permettere di raggiungere zone anche molto distanti, viceversa, con la batteria quasi esaurita deve cercare di visitare luoghi più vicini in modo da conservare la carica fino alla fine dell'esplorazione.

[3] È possibile trovare questo file nella cartella `System` all'interno della directory principale dell'ambiente di lavoro.

4.5 Ulteriori strategie per il confronto delle prestazioni

Oltre alle varie versioni della strategia basata su MCDM, abbiamo deciso di implementare altre due semplici strategie, in modo da avere altre tecniche, da aggiungere a quella ricavata dall'applicazione del codice originale di AOJRF, con cui confrontare le prestazioni del nostro approccio.
La prima strategia, nonché la più semplice, consiste nell'associare ad ogni frontiera f un'utilità calcolata tramite la formula:

$$u(f) = d(f)$$

In questo caso, quindi, l'utilità di un punto dipende esclusivamente dalla sua distanza $d(f)$ rispetto al robot in esame. Di conseguenza ci aspettiamo che ogni agente preferisca visitare zone più vicine al luogo in cui si trova, indipendentemente dall'area che potrebbe esplorare una volta giunto in tale posizione, piuttosto che percorrere grandi distanze. Nell'inserire tale strategia all'interno del codice, abbiamo deciso di mantenere il valore della distanza normalizzato, tramite la stessa formula utilizzata nella prima versione del nostro software, in modo da non dover modificare tutti i controlli riguardanti la scelta dell'utilità massima. Se non avessimo fatto questa scelta, infatti, la migliore utilità sarebbe stata quella con valore minore rispetto a tutte quelle calcolate per ogni coppia (robot, frontiera).
Per quanto riguarda la seconda strategia, abbiamo valutato l'utilità di ogni frontiera f utilizzando una formula studiata appositamente per risolvere il problema dell'esplorazione di ambienti sconosciuti da parte di squadre di robot [5, 6] che, oltre alla distanza $d(f)$ contempla anche la stima dell'area $A(f)$ presente al di là della frontiera in esame:

$$u(f) = A(f) - \beta \cdot d(f)$$

Tale formula rappresenta una sorta di somma pesata tra la distanza del robot dalla frontiera e la stima dell'area. Qui β è un elemento che determina

l'importanza relativa dell'area stimata rispetto alla distanza e che può assumere valori compresi nell'intervallo $[0.01, 50]$. Noi abbiamo sempre impostato $\beta = 1$ in quanto è il valore utilizzato dagli sviluppatori della strategia che, inoltre, hanno dimostrato, tramite esperimenti, che le prestazioni non variano di molto scegliendo un diverso valore di β. Visto che si tratta di una formula studiata *ad hoc* bisogna notare che, in questo caso, i criteri non sono stati normalizzati.
Anche delle due strategie appena introdotte, che abbiamo chiamato *DISTANZA* e *SOMMA PESATA*, abbiamo sviluppato, come nel caso della nostra e di quella di AOJRF, le due diverse versioni. Queste differiscono solo per quanto riguarda il punto da raggiungere all'interno dell'area associata alla frontiera che il robot sceglie come suo obiettivo ed è possibile trovarle all'interno del DVD allegato rispettivamente nei file `DISTANZA V.1`, `DISTANZA V.2`, `SOMMA PESATA V.1` e `SOMMA PESATA V.2`.

Capitolo 5

Realizzazioni sperimentali e valutazioni

La fase sperimentale del nostro lavoro è stata svolta per dimostrare i punti di forza e verificare le potenzialità di una strategia basata su MCDM.
Nell'esecuzione degli esperimenti abbiamo utilizzato due mappe raffiguranti due ambienti con caratteristiche molto diverse tra loro. Quella mostrata in Figura 5.1 è la rappresentazione di un piano di un hotel ed è una delle mappe utilizzate nella RoboCup Rescue Virtual Robot Competition del 2006. Questo ambiente è composto da un corridoio verticale al centro, che connet-

Figura 5.1: Mappa raffigurante un piano di un hotel

te l'ingresso con diversi corridoi che si diramano a sinistra e a destra, e da numerose stanze. L'hotel si presta perfettamente al nostro scopo in quanto l'esplorazione in un ambiente simile è particolarmente difficile dato il modo in cui è strutturato, ossia per la presenza, al suo interno, di vittime e di molti ostacoli, quali muri, scrivanie, ecc.... Dopo aver effettuato alcuni esperimenti su questa mappa abbiamo pensato di eseguirne altri in uno spazio aperto per poter esaminare il comportamento della nostra strategia in condizioni completamente diverse. Tra le mappe disponibili in USARSim, non siamo stati in grado di sfruttare quella più adatta per questo tipo di simulazione, a causa delle sue grandi dimensioni e dell'enorme consumo di memoria. Per ovviare a questo problema abbiamo scelto l'ambiente mostrato in Figura 5.2, e abbiamo cercato di posizionare gli agenti che eseguono l'esplorazione nei pressi dell'area libera da ostacoli situata in basso a destra nella mappa, in cui, al contrario dell'ambiente precedente, sono presenti pochi ostacoli e le vittime sono completamente assenti.

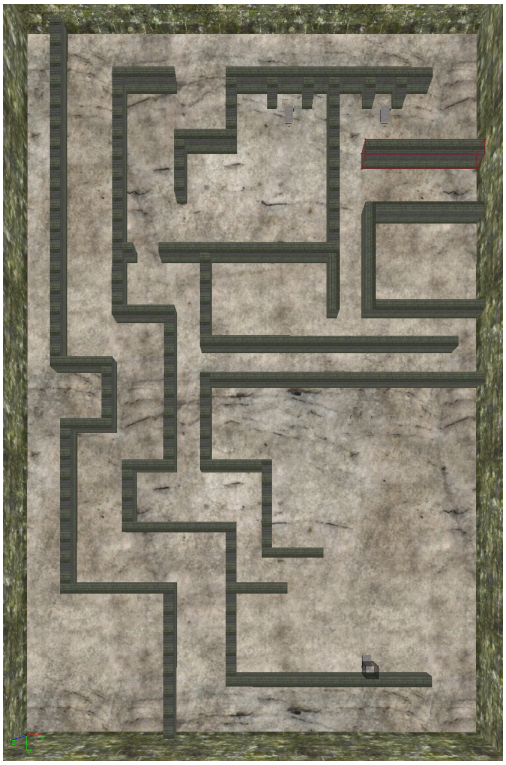


Figura 5.2: Mappa raffigurante lo spazio aperto

Al fine di ottenere una quantità di dati sufficiente per poter trarre delle conclusioni interessanti abbiamo effettuato un totale di duecento prove, ognuna della durata di venti minuti, come accade durante la RoboCup Rescue Virtual Robot Competition. A questo scopo abbiamo utilizzato, oltre alla nostra strategia, anche le altre menzionate nel Capitolo 4 per poterne confrontare i risultati. Da ora in poi, a questi metodi di esplorazione ci riferiremo con i nomi di POLIMI, DISTANZA, SOMMA PESATA e AOJRF.
I robot, esplorando l'ambiente, rilevano, grazie all'utilizzo di tre diversi sensori di cui sono forniti, tre differenti tipi di area, due delle quali sono già state citate nella Sezione 2.3 e nella Sezione 4.1.2 relativamente alla definizione delle frontiere:

- Free area: spazio considerato libero da ostacoli osservato, tramite un sensore di distanza, entro un raggio massimo di 20 metri dal robot e colorato di bianco nella mappa risultante dall'esplorazione.

- Safe area: spazio libero da ostacoli che rappresenta l'area effettivamente esplorata, utilizzando un sensore di distanza più accurato, entro un raggio massimo di 3 metri dal robot e colorata di grigio nella mappa risultante dall'esplorazione.

- Clear area: spazio libero da ostacoli e da vittime, osservato tramite un sensore che rileva la presenza di queste ultime, e colorato di verde nella mappa risultante dall'esplorazione. Questa misura non differisce molto da quella precedente.

Allo scadere di ogni minuto di esecuzione dell'esplorazione abbiamo fatto in modo di ottenere i valori relativi a queste grandezze. In questo modo abbiamo potuto considerare l'andamento dei tre tipi di area, espresso in m^2, rispetto al tempo, per valutare e confrontare le prestazioni delle diverse strategie.
Gli esperimenti che abbiamo effettuato sono così ripartiti:

- Al fine di valutare la prima versione di POLIMI, abbiamo effettuato i seguenti gruppi di esperimenti, ognuno dei quali ripetuto quattro volte, per le quattro strategie considerate:

  - Dieci esperimenti relativi a dieci diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e due P2AT, nell'ambiente virtuale di Figura 5.1.
  - Dieci esperimenti relativi a dieci diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e un P2AT, nell'ambiente virtuale di Figura 5.1.
  - Dieci esperimenti relativi a dieci diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e due P2AT, nell'ambiente virtuale di Figura 5.2.
  - Dieci esperimenti relativi a dieci diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e un P2AT, nell'ambiente virtuale di Figura 5.2.

- Allo scopo di controllare i risultati ottenuti dopo aver apportato la modifica riguardante la posizione che un robot deve raggiungere all'interno dell'area relativa alla frontiera scelta, descritta nella Sezione 4.3, abbiamo effettuato i seguenti esperimenti, per ognuno dei quali abbiamo svolto quattro prove, una per ogni strategia considerata:

  - Cinque esperimenti relativi a cinque diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e un P2AT, nell'ambiente virtuale di Figura 5.2.

- Per verificare il comportamento della nostra strategia, basata su MCDM, abbiamo effettuato i seguenti gruppi di esperimenti, per ognuno dei quali abbiamo svolto due prove, una per ogni soluzione trovata e descritta nella Sezione 4.4 relativamente all'introduzione della batteria tra le informazioni da considerare per la valutazione di un punto:

  - Cinque esperimenti relativi a cinque diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e due P2AT, nell'ambiente virtuale di Figura 5.1.

– Cinque esperimenti relativi a cinque diverse posizioni iniziali, eseguiti con una squadra composta da una ComStation e due P2AT, nell'ambiente virtuale di Figura 5.2.

I risultati dei nostri esperimenti sono mostrati e discussi nelle sezioni successive, e, per ognuno di questi, è possibile trovare, all'interno della cartella `Esperimenti` nel DVD allegato, i dati relativi ai tre tipi di area rilevati, i grafici che mostrano il loro andamento rispetto al tempo e il confronto tra le mappe risultanti dall'applicazione delle diverse strategie. Le immagini che raffigurano questo confronto tra le mappe sono due, una in cui sono state visualizzate, oltre al percorso seguito dai robot e alle posizioni di ostacoli e vittime, solo le aree free e safe, l'altra comprendente anche l'area clear.
Per dare a chiunque la possibilità di ripetere le stesse prove da noi effettuate abbiamo riportato nell'Appendice B la descrizione di una di queste unitamente a tutte le impostazioni adottate, mentre nell'Appendice C abbiamo elencato tutte le posizioni iniziali dei robot per ogni esperimento.

5.1 Studio dei primi risultati ottenuti

Durante la prima sessione di test abbiamo eseguito un totale di centosessanta prove che differiscono tra loro per il numero dei componenti della squadra, l'ambiente in cui si svolgono e il metodo di esplorazione utilizzato. In questa fase, in cui abbiamo preso in considerazione la prima versione di tutte le strategie esaminate, ci aspettiamo di ottenere con la nostra strategia risultati, se non migliori, almeno paragonabili a quelli di AOJRF e SOMMA PESATA, e nettamente più soddisfacenti di quelli relativi all'applicazione di DISTANZA.

Abbiamo eseguito i primi dieci esperimenti all'interno dell'hotel, con una squadra di agenti composta da una ComStation e due robot P2AT, aventi il compito di esplorare l'ambiente. Agli esperimenti abbiamo associato diversi insiemi di posizioni iniziali, per i componenti della squadra. In questo modo abbiamo potuto eseguire quattro prove per ogni esperimento, modificando solo la strategia, ma mantenendo le stesse condizioni di partenza.

In tutti i casi le posizioni iniziali sono state scelte generando numeri casuali, per le coordinate x e y, controllando che queste non corrispondessero a punti esterni alla mappa, coincidenti con ostacoli o vittime o luoghi da cui il robot non potesse muoversi e, contemporaneamente, cercando di coprire l'intero ambiente.
Nella Tabella 5.1 sono riportati i valori relativi all'area safe risultanti al termine di ogni test, mentre la Tabella 5.2 contiene quelli relativi all'area clear. In entrambe le tabelle è indicato in grassetto, per ogni esperimento, il valore di area più elevato, che indica, quindi, la strategia che ha permesso di esplorare una maggiore porzione di ambiente, relativamente alle due aree. Nei grafici in Figura 5.3 viene, invece, mostrato l'andamento medio di ogni strategia, fatto sui dieci esperimenti, per i due tipi di area, rispetto al tempo di esecuzione. Come si può subito notare, DISTANZA ha confermato le nostre previsioni confermandosi la strategia peggiore, ossia quella che ha visitato una minore quantità di ambiente. Tra le altre, nei singoli esperimenti, SOMMA PESATA si è comportata spesso peggio, mentre possiamo essere soddisfatti delle prestazioni di POLIMI in quanto, con la sua applicazione, i robot esplorano quantità di area a volte superiori anche di quelle esplorate con AOJRF. Il comportamento medio di queste tre strategie, comunque, si equivale.

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	417.64	**420.52**	185.33	327.04
2	**405.33**	402.25	252.29	362.12
3	**462.72**	420.92	264.22	385.03
4	**475.00**	417.53	267.63	411.98
5	**539.11**	457.26	156.26	502.14
6	379.56	**439.33**	213.10	387.45
7	436.72	**443.91**	380.69	322.87
8	496.47	463.00	217.18	**560.24**
9	408.64	419.09	244.52	**437.15**
10	**500.00**	369.93	233.99	469.08

Tabella 5.1: Valori dell'area safe ricavati con due P2AT nell'hotel

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	343.60	**371.71**	170.84	274.42
2	328.93	**332.58**	229.89	321.04
3	**357.80**	357.28	244.69	323.42
4	**398.65**	359.52	263.97	348.32
5	**445.38**	349.74	144.19	415.63
6	321.12	**351.47**	206.84	342.66
7	357.22	**374.83**	366.96	263.40
8	392.85	375.09	210.23	**450.77**
9	351.24	331.31	209.34	**370.99**
10	394.08	331.88	213.32	**401.51**

Tabella 5.2: Valori dell'area clear ricavati con due P2AT nell'hotel

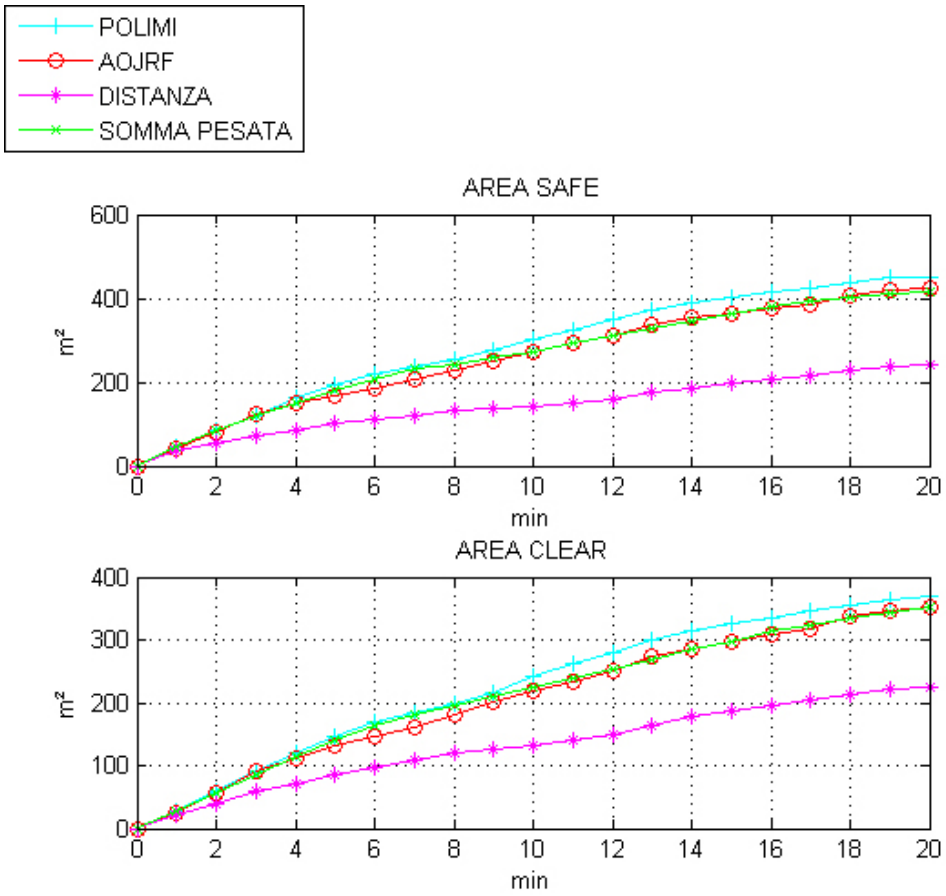


Figura 5.3: Andamento medio delle strategie con due P2AT nell'hotel

Non abbiamo riportato i valori relativi all'area free in quanto, valutando i risultati, abbiamo notato che, per il confronto delle prestazioni dei diversi metodi di esplorazione, non ci possiamo basare su questa grandezza a causa della possibilità di incorrere in situazioni particolari. Quanto detto è dimostrato dai risultati dell'esperimento 7, per cui i valori sono riportati in Tabella 5.3 e le mappe create al termine dell'esplorazione sono mostrate nella Figura 5.4, relativamente alle sole aree free e safe, e nella Figura 5.5, che riproduce anche la porzione di ambiente relativa all'area clear.

AREA	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
FREE	652.05	648.35	**1733.09**	580.92
SAFE	436.72	**443.91**	380.69	322.87
CLEAR	357.22	**374.83**	366.96	263.40

Tabella 5.3: Valori di area risultanti dall'esperimento 7

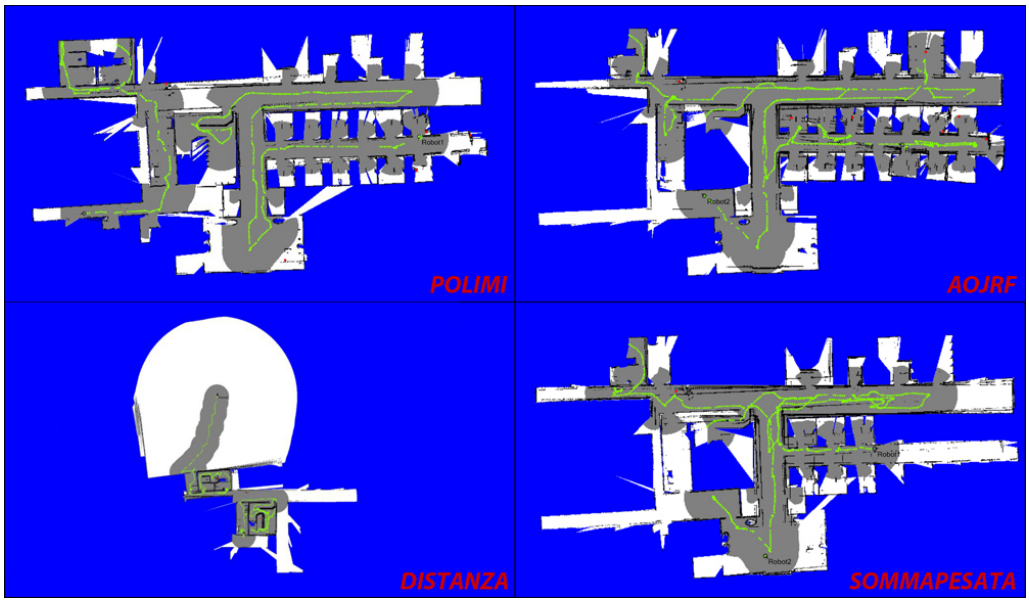


Figura 5.4: Mappe risultanti dall'esperimento 7 relative alle aree free e safe

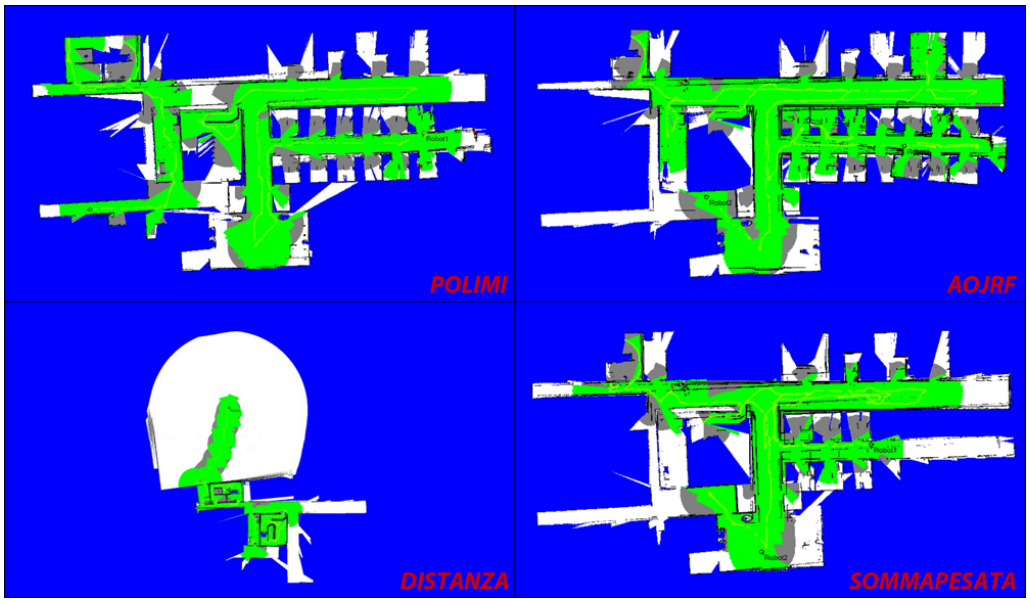


Figura 5.5: Mappe risultanti dall'esperimento 7 relative alle aree free, safe e clear

In un caso del genere si può notare che l'area free relativa a DISTANZA assume un valore molto elevato, e che questo è dovuto solamente al fatto che un robot entra in un ampio cortile. Sia l'area effettivamente esplorata che quella "vista" dal sensore che rileva la presenza di vittime, infatti, non sono le migliori e comunque non presentano la stessa differenza che esiste tra il valore dell'area free rilevato con DISTANZA e quelli delle altre strategie.
Durante l'esecuzione degli esperimenti in queste condizioni, ovvero nell'ambiente considerato e con una squadra di robot composta da una ComStation e due P2AT, ci siamo imbattuti in situazioni che ci hanno costretto a ripetere alcuni test dall'inizio o ad intervenire sui movimenti dei robot. In alcuni casi è capitato che un robot impiegasse molto tempo per scegliere il prossimo punto da raggiungere e che, piuttosto di fermarsi, durante l'elaborazione, proseguisse dritto davanti a sé andando a sbattere contro un muro e ribaltandosi. In tali situazioni, che abbiamo notato verificarsi solitamente verso la fine dell'esplorazione e principalmente in presenza di un alto numero di frontiere da valutare, poichè è impossibile modificare il controllore del robot per fermarne il movimento prima che abbia finito di "pensare", siamo stati costretti a ripetere l'intero test. Altre volte, invece, è successo che un ro-

bot restasse bloccato sotto un tavolo o che sbattesse contro la gamba di una scrivania e che non riuscisse più a procedere lungo il percorso stabilito. Per ovviare a questo tipo di situazioni siamo potuti intervenire attivando Tele Operation, per portare manualmente il robot in una posizione attigua in cui potesse muoversi liberamente, e successivamente riavviando la strategia precedente, senza dover ripetere il test. Questo è successo con tutte le strategie, ma in particolar modo con DISTANZA in quanto, visto che il robot è portato a visitare zone a lui molto vicine, indipendentemente dalla loro dimensione, in un ambiente come quello utilizzato, è più probabile che si debba muovere in piccoli spazi occupati da ostacoli. Un esempio significativo è mostrato dalle mappe di Figura 5.6, relative all'esperimento 10 di cui ci interessa mostrare il percorso del robot. Questo, come si può vedere, al contrario di quello delle altre strategie, che è più regolare, nel caso di DISTANZA è più articolato e si snoda nei luoghi più remoti di ogni stanza visitata. In questo esperimento, mentre gli interventi manuali per "sbloccare" i robot con POLIMI, AOJRF e SOMMA PESATA non sono stati più di due, con DISTANZA ne abbiamo effettuati quattro volte tanto.

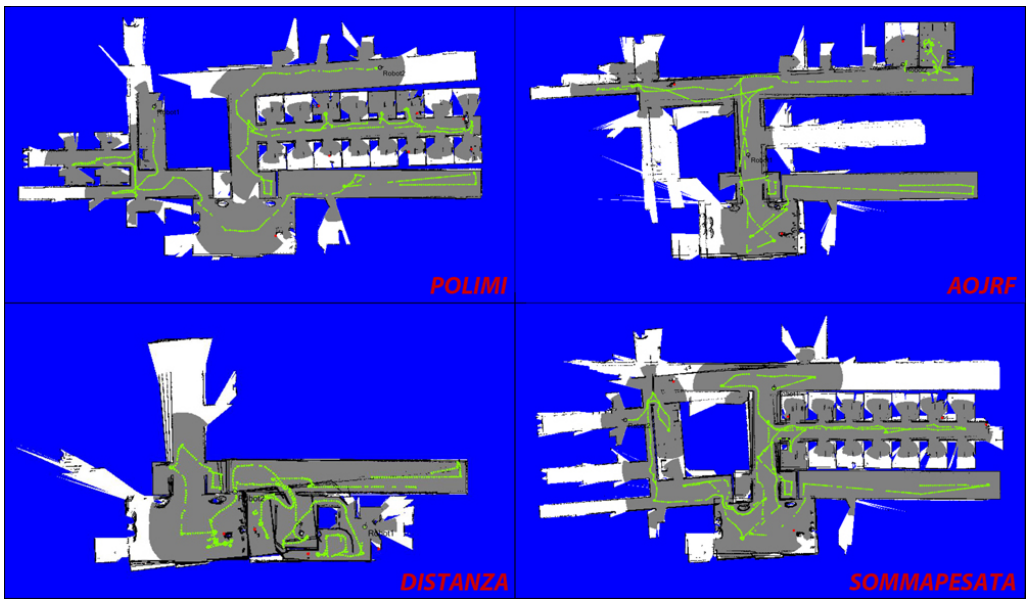


Figura 5.6: Mappe risultanti dall'esperimento 10

Infine vogliamo sottolineare il fatto che, guardando i grafici che mostrano, per ogni singolo esperimento, l'andamento delle aree rispetto al tempo, tutti presenti nel DVD allegato, si possono notare tratti quasi orizzontali. Questi stanno a significare che, nell'arco di tempo che sottendono, gli agenti non hanno acquisito area significativa. Anche questo fatto si verifica particolarmente con DISTANZA e meno con le altre strategie. A titolo d'esempio riportiamo, nella figura Figura 5.7, il grafico dell'esperimento 3 relativo alla sola area safe, ma ciò si può notare anche in altri esperimenti.

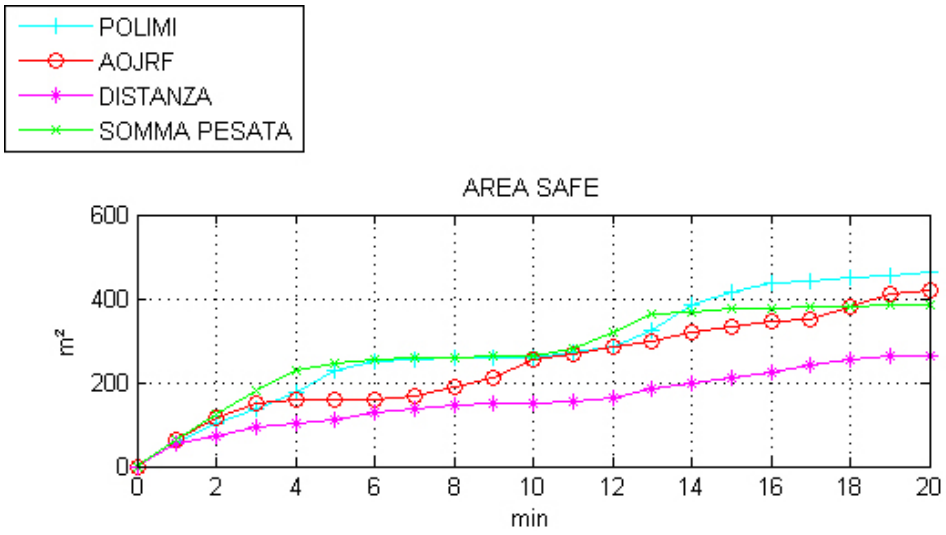


Figura 5.7: Andamento dell'area safe dell'esperimento 3

Successivamente abbiamo deciso di provare ad effettuare ulteriori esperimenti modificando il numero di componenti della squadra. Ne abbiamo quindi eseguiti altri dieci, all'interno dell'hotel, utilizzando una ComStation e un solo robot P2AT e basandoci su quelli già fatti nello stesso ambiente per la scelta delle posizioni iniziali. Per ognuno di essi abbiamo mantenuto le stesse coordinate per la ComStation, mentre per il P2AT abbiamo effettuato una scelta tra quelle associate ai due robot precedentemente, cercando di avere posizioni iniziali abbastanza diverse in ogni esperimento.
Inizialmente l'idea era quella di utilizzare una ComStation e tre robot per l'esplorazione, ma si è rivelata irrealizzabile, a causa dell'eccessiva richiesta

di memoria. Come con gli altri esperimenti eseguiti nell'hotel e appena descritti, anche in questo caso ci è capitato che i robot urtassero contro il muro e che il programma non rispondesse ai comandi per permetterci di intervenire e fermarli in tempo. La differenza sostanziale, però, è che con una squadra di agenti così composta non siamo mai riusciti ad effettuare una prova completa, ma soltanto pochi minuti di esplorazione. Con l'utilizzo di un solo P2AT, al contrario, non abbiamo avuto alcun problema di questo tipo, ma ci è capitato solamente di dover intervenire per spostare robot rimasti bloccati lungo il percorso. Anche con la modifica del numero di agenti, per gli stessi motivi descritti in precedenza, questo fatto si è verificato maggiormente con DISTANZA piuttosto che con le altre strategie.
Come prima, riportiamo all'interno delle Tabelle 5.4 e 5.5 rispettivamente i valori relativi all'area safe e all'area clear risultanti al termine di ogni prova e nella Figura 5.8 l'andamento medio di queste rispetto al tempo. Confrontando questi risultati con quelli degli esperimenti precedenti il primo aspetto da sottolineare è che la quantità di area mappata da due P2AT è quasi il doppio rispetto a quella risultante dall'esplorazione effettuata da un solo P2AT. Questo è dovuto sia alla differenza del numero di robot impiegati che alla presenza della comunicazione tra i due agenti che permette loro di esplorare zone della mappa diverse e non ancora visitate.

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**304.52**	256.71	142.90	274.98
2	362.67	277.79	135.48	**376.21**
3	242.33	**279.69**	128.52	252.01
4	**306.71**	292.71	90.40	299.76
5	**316.22**	273.14	168.43	300.16
6	**353.20**	254.49	117.86	296.56
7	256.36	**313.80**	214.49	266.52
8	319.60	306.63	155.17	**330.87**
9	325.64	**338.76**	130.05	318.70
10	241.33	202.00	157.34	**295.65**

Tabella 5.4: Valori dell'area safe ricavati con un P2AT nell'hotel

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**277.92**	232.47	133.20	240.01
2	279.20	227.91	129.18	**281.95**
3	212.64	**249.88**	113.88	212.93
4	**250.17**	244.05	85.61	238.17
5	**264.39**	238.90	151.97	252.53
6	**280.30**	235.12	112.08	225.31
7	220.54	**261.24**	179.87	216.23
8	250.04	240.60	143.43	**273.12**
9	248.34	**278.01**	128.64	244.71
10	219.70	183.99	157.17	**236.34**

Tabella 5.5: Valori dell'area clear ricavati con un P2AT nell'hotel

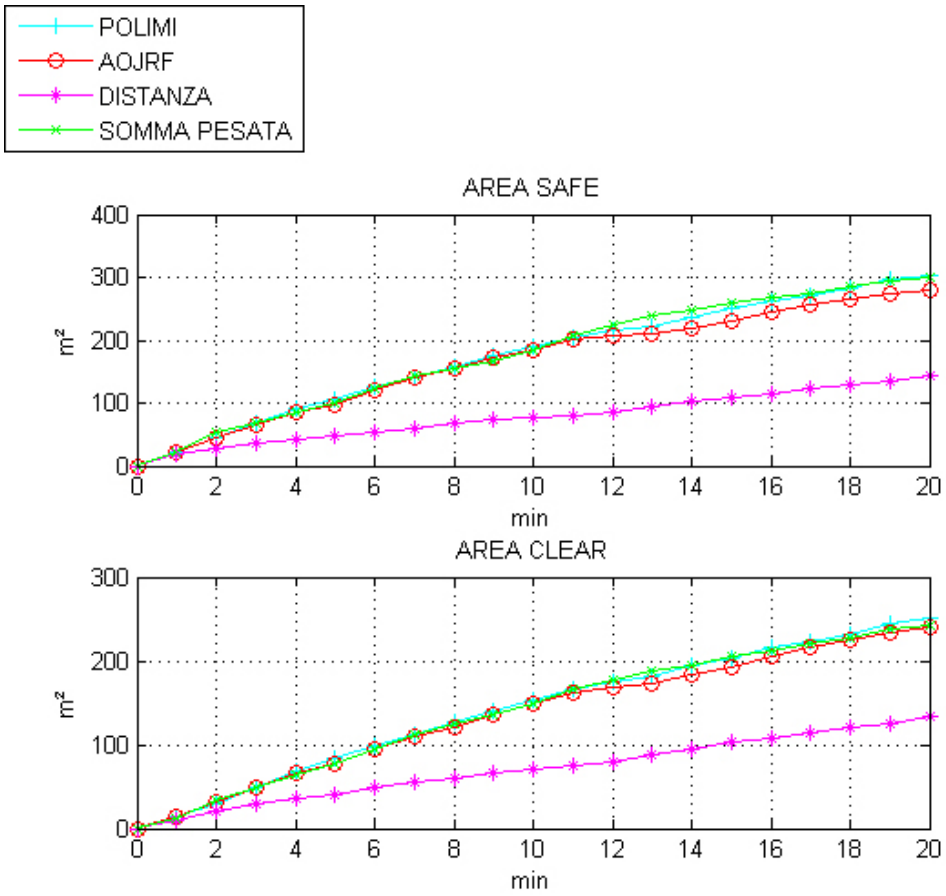


Figura 5.8: Andamento medio delle strategie con un P2AT nell'hotel

Detto questo, dai risultati di questi esperimenti possiamo trarre le stesse conclusioni di quelli ottenuti con l'utilizzo di due robot P2AT, per quanto riguarda la quantità di area esplorata, con l'unica eccezione di SOMMA PESATA le cui prestazioni sembrano migliorate e paragonabili a quelle di POLIMI e AOJRF. Inoltre, un altro aspetto che accomuna i due insiemi di esperimenti è la presenza, anche in questo caso, nei grafici relativi ad ogni esperimento, di tratti orizzontali nell'andamento delle aree rispetto al tempo. Anche qui ciò succede in particolar modo con l'applicazione di DISTANZA.

Dopo questi esperimenti, tornando ad utilizzare una squadra composta da tre agenti, ossia due P2AT e una ComStation, ne abbiamo eseguiti altri dieci cambiando solamente l'ambiente da esplorare. Di conseguenza abbiamo dovuto determinare nuove posizioni iniziali, e l'abbiamo fatto seguendo esattamente la stessa procedura adottata precedentemente, ossia scegliendole casualmente e cercando di coprire tutta la zona dell'ambiente che abbiamo utilizzato. La nuova mappa, come già accennato, rappresenta uno spazio aperto privo di vittime e con la presenza di un limitato numero di ostacoli. Anche qui le Tabelle 5.6 e 5.7 e la Figura 5.9 mostrano i risultati ottenuti relativamente ai valori finali delle aree safe e clear e al loro andamento medio rispetto al tempo.

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**720.64**	623.86	679.03	635.10
2	**913.34**	677.38	651.83	896.32
3	752.13	602.82	**781.76**	753.24
4	**919.03**	851.21	769.23	814.30
5	**726.05**	715.06	595.89	704.04
6	**965.54**	741.57	790.42	869.17
7	**884.79**	755.71	703.88	853.57
8	**946.48**	642.33	697.63	774.17
9	**913.86**	878.12	524.24	742.79
10	770.92	**786.94**	546.83	736.06

Tabella 5.6: Valori dell'area safe ricavati con due P2AT nello spazio aperto

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**612.11**	527.38	600.03	520.04
2	**772.36**	576.53	591.14	752.51
3	633.56	499.01	**651.22**	595.97
4	**764.45**	736.33	666.52	687.06
5	573.31	586.07	516.29	**608.04**
6	**787.64**	616.06	716.73	693.88
7	692.07	620.64	643.34	**695.59**
8	**702.16**	582.04	596.68	636.48
9	**762.82**	703.75	455.01	631.93
10	627.18	**659.33**	465.81	632.33

Tabella 5.7: Valori dell'area clear ricavati con due P2AT nello spazio aperto

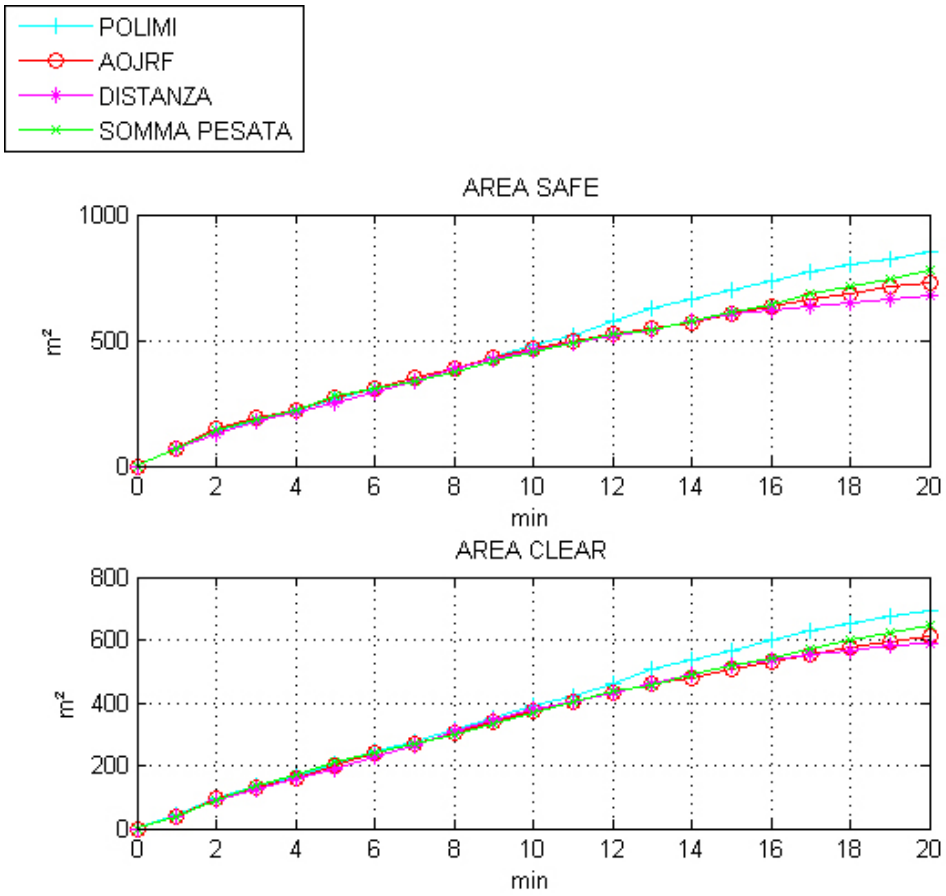


Figura 5.9: Andamento medio delle strategie con due P2AT nello spazio aperto

Anche durante l'esplorazione all'interno dello spazio aperto siamo dovuti intervenire manualmente, ma non per i motivi che ci hanno portato a farlo nel caso dell'hotel. Qui abbiamo riscontrato un diverso problema, già discusso nella Sezione 4.3, dovuto alla scelta del punto da raggiungere all'interno dell'area presente al di là della frontiera selezionata, dipendente non dalla strategia utilizzata ma da come è stato progettato il software originale. In questo caso, infatti, non c'è stata una strategia che ha richiesto un maggior numero di interventi rispetto alle altre, e per ognuna abbiamo dovuto interferire nell'esecuzione una media di tre volte a esperimento.
Dai risultati si nota che POLIMI ha esplorato spesso una maggiore quantità di area ma che, comunque, tutte le strategie hanno prestazioni simili. Ciò è dovuto al tipo di ambiente considerato, che tende ad appiattire le differenze di comportamento dei diversi metodi di esplorazione. Questo accade in quanto, ogni volta che un robot deve decidere quale sarà il suo prossimo obiettivo, il numero di frontiere tra cui scegliere, presenti nello spazio aperto, è minore rispetto a quello di un ambiente più complesso, come l'hotel. Quindi la scelta di una frontiera nell'hotel è più difficile e fa emergere maggiormente le differenze fra le strategie. Per verificare quanto appena detto abbiamo calcolato il numero medio di frontiere presenti nei due ambienti, rispetto al numero totale di decisioni prese dai robot, durante i venti minuti che compongono una prova. Questa media è stata ottenuta sommando il numero di frontiere tra cui deve scegliere ogni robot ad ogni passo di esplorazione e dividendo questo valore per la somma delle volte in cui ogni robot deve prendere una decisione:

$$Numero\ Medio\ Di\ Frontiere\ Nell'Hotel = 40.55$$
$$Numero\ Medio\ Di\ Frontiere\ Nello\ Spazio\ Aperto = 8.42$$

Il risultato ottenuto conferma il fatto che nello spazio aperto, in un dato passo di esplorazione, sono presenti poche frontiere, rendendo più probabile che le diverse strategie scelgano lo stesso obiettivo e facendo sì che tutte abbiano comportamenti simili.
Anche per il confronto delle prestazioni delle strategie relativo a questo nuovo

ambiente non abbiamo considerato l'area free in quanto spesso succede che il sensore ad essa relativo copra tutta la zona esplorabile, non molto vasta rispetto al suo raggio d'azione, pochi minuti dopo l'inizio dell'esplorazione. Un esempio di tale situazione si può notare dal grafico in Figura 5.10, che mostra l'andamento dell'area free rispetto al tempo, risultante dall'esperimento 1.

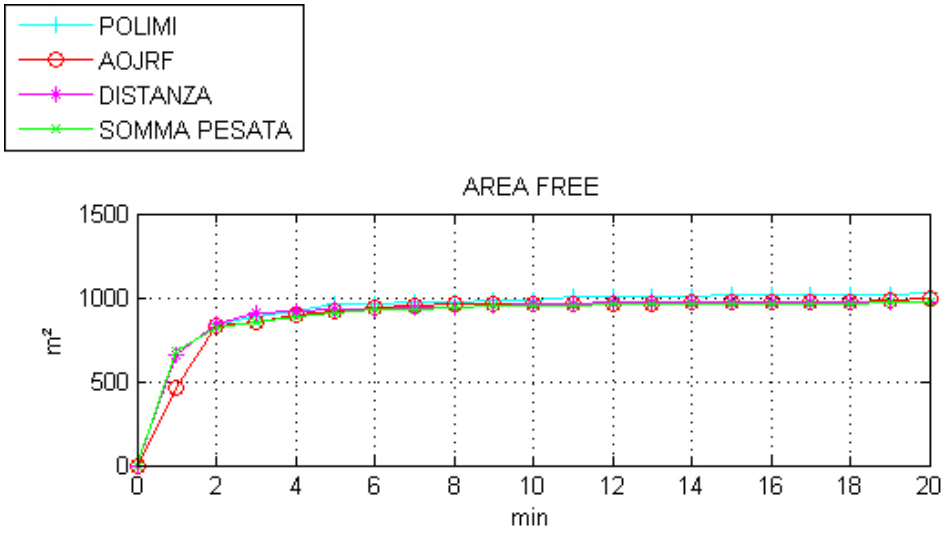


Figura 5.10: Andamento dell'area free dell'esperimento 1

Esattamente come con gli esperimenti svolti nell'hotel, anche nel caso dello spazio aperto abbiamo deciso di effettuarne altri dieci con una squadra di robot composta non più da tre ma da due agenti, ossia un P2AT e una ComStation, e, come prima, per la scelta delle loro posizioni iniziali ci siamo basati su quelle definite per gli esperimenti già eseguiti nello stesso ambiente. Anche durante queste prove abbiamo dovuto modificare il percorso del P2AT per risolvere il già discusso problema che si verifica solo nello spazio aperto. L'unica differenza è che, data la presenza di un solo robot per l'esplorazione invece di due, in ogni esperimento siamo dovuti intervenire non più di tre volte per strategia.
All'interno delle Tabelle 5.8 e 5.9 riportiamo i valori relativi alle aree safe e clear risultanti al termine di ogni prova e nella Figura 5.11 il loro andamento

medio rispetto al tempo.
Dallo studio di questi risultati possiamo confermare quanto già detto a proposito degli esperimenti svolti nello stesso ambiente ma con un componente in più nella squadra di agenti. Anche qui POLIMI si conferma la strategia con cui viene esplorata più area, sebbene anche le altre abbiano buone prestazioni, e vale lo stesso discorso fatto per quanto riguarda l'appiattimento delle differenze di comportamento. Considerando entrambi gli ambienti e calcolando il numero medio di frontiere presenti durante i venti minuti di esplorazione abbiamo, infatti, ottenuto:

$$Numero\ Medio\ Di\ Frontiere\ Nell'Hotel = 29.55$$
$$Numero\ Medio\ Di\ Frontiere\ Nello\ Spazio\ Aperto = 4.83$$

Questi valori sono stati calcolati utilizzando lo stesso procedimento adottato precedentemente nel caso di due robot esploratori. Come si può notare, il numero medio di frontiere presente con un solo P2AT vale circa la metà rispetto a quello ottenuto nel caso di un'esplorazione effettuata con due P2AT. Questi risultati ci permettono, quindi, di trarre le stesse conclusioni discusse nel caso precedente.

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**574.50**	479.11	411.89	532.73
2	**591.27**	540.87	475.24	471.56
3	526.28	517.61	482.67	**545.40**
4	539.82	512.60	532.68	**575.55**
5	630.49	**657.35**	345.29	607.07
6	**660.80**	518.24	502.84	445.43
7	**631.01**	581.99	489.14	577.29
8	**559.83**	501.00	468.79	450.02
9	**556.36**	510.04	429.39	495.06
10	**559.64**	444.33	491.80	484.01

Tabella 5.8: Valori dell'area safe ricavati con un P2AT nello spazio aperto

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	441.22	415.50	349.04	**444.91**
2	**463.29**	423.43	404.40	393.22
3	411.64	**433.16**	398.71	411.27
4	**447.56**	419.53	429.55	422.80
5	456.80	**484.32**	316.56	450.68
6	**491.03**	434.32	433.07	388.65
7	**454.71**	440.36	418.59	435.07
8	**474.30**	430.47	394.89	360.78
9	**446.00**	420.82	349.27	417.20
10	**445.60**	401.29	410.42	418.47

Tabella 5.9: Valori dell'area clear ricavati con un P2AT nello spazio aperto

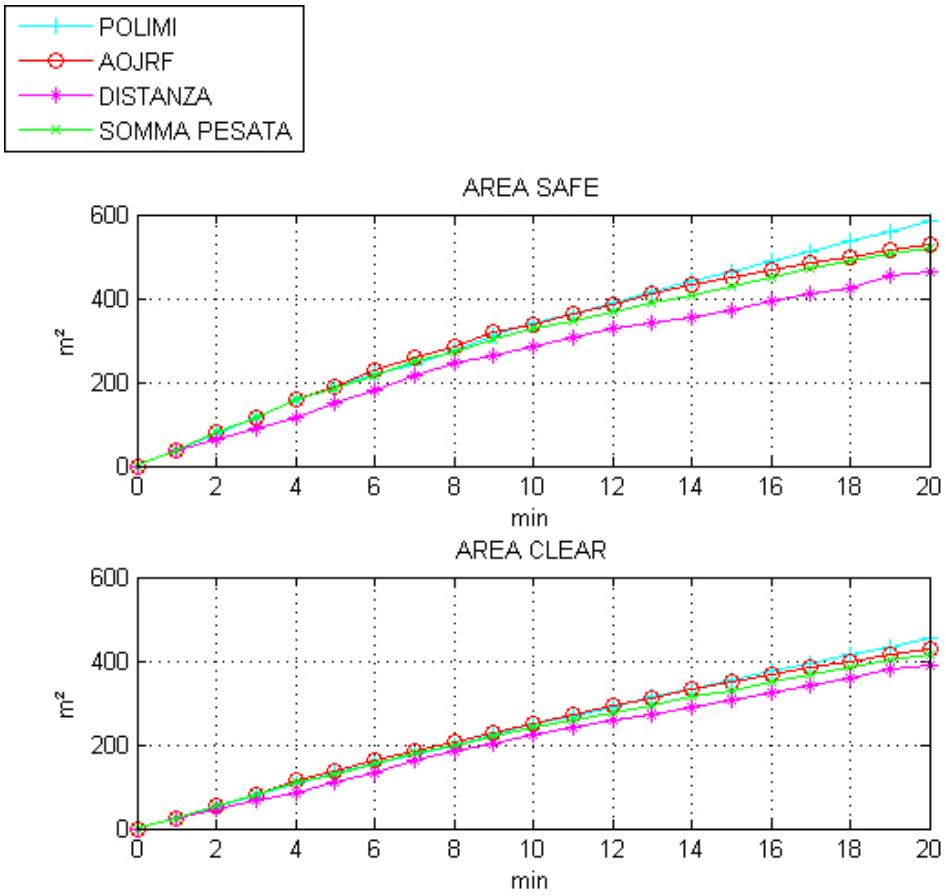


Figura 5.11: Andamento medio delle strategie con un P2AT nello spazio aperto

5.1.1 Valutazione statistica dei risultati

Le conclusioni che abbiamo tratto fin'ora, basandoci sui risultati di queste prime centosessanta prove, ci portano ad affermare che POLIMI, AOJRF e SOMMA PESATA hanno comportamenti simili tra loro e diversi da quello di DISTANZA che, tra tutte è la strategia peggiore. Questo fatto, però, si verifica solo in un ambiente con molte stanze, corridoi e ostacoli, ossia con caratteristiche simili a quelle possedute dall'ambiente virtuale che rappresenta il piano di un hotel e che abbiamo utilizzato durante i primi esperimenti. In uno spazio aperto, al contrario, sembra che tutti i metodi di esplorazione analizzati abbiano le stesse prestazioni. Per confermare l'esattezza di queste supposizioni, abbiamo eseguito un'analisi statistica sugli insiemi di dati ricavati dalle prove effettuate, per ogni gruppo di dieci esperimenti, in modo da verificare l'esistenza di una differenza statisticamente significativa tra le strategie.
Per raggiungere il nostro scopo abbiamo utilizzato un test di analisi della varianza, o *ANOVA (ANalysis Of VAriance)* [17] che permette di confrontare due o più serie di campioni e stabilire se questi hanno o meno la stessa media, verificando la validità dell'ipotesi nulla, H_0, che le medie siano uguali per ogni serie, contro l'ipotesi alternativa, H_1, che almeno due medie siano tra loro differenti. Per fare ciò utilizza una statistica T, data dal rapporto tra lo stimatore non distorto della varianza tra i gruppi (detta between) e quello della varianza interna ai gruppi (detta within). Se:

$$T > F(\alpha, k-1, k \cdot (n_i - 1))$$

dove k e n_i indicano rispettivamente il numero dei gruppi e la numerosità di ogni singolo gruppo e $F(\alpha, k-1, k \cdot (n-1))$ rappresenta il valore della distribuzione di Fisher con $k-1$ e $k \cdot (n_i - 1)$ gradi di libertà a livello di significatività α, allora si rifiuta l'ipotesi H_0, altrimenti questa viene accettata.
Utilizzando ANOVA per il confronto di due serie di dati si ottengono gli stessi risultati che si avrebbero dall'applicazione del test t di Student [9], in quanto ANOVA ne è una generalizzazione. In presenza di più insiemi di dati, però, il loro confronto a due a due attraverso il t-test aumenterebbe le possibilità di

commettere un errore di primo tipo, ossia di rifiutare l'ipotesi nulla quando, invece, è vera.
La Tabella 5.10 mostra i risultati che abbiamo ottenuto eseguendo, quindi, il test ANOVA. Per farlo abbiamo considerato separatamente gli esperimenti in base all'ambiente e alla squadra di agenti utilizzati e abbiamo applicato il test statistico una volta per valutare l'area safe e una per l'area clear. Gli insiemi di campioni, contenenti ognuno i dieci valori relativi all'area, rilevati al termine del ventesimo minuto di esplorazione, e coinvolti nell'analisi della varianza, sono quattro ogni volta, uno per ogni strategia.

	T SAFE	T CLEAR
2 P2AT HOTEL	28.84	19.72
1 P2AT HOTEL	41.05	49.85
2 P2AT SPAZIO APERTO	6.76	3.47
1 P2AT SPAZIO APERTO	8.36	8.74

Tabella 5.10: Risultati del test ANOVA

In tutti i casi risulta:

$$T > F(5\%, 3, 36) = 2.92$$

e di conseguenza rifiutiamo sempre l'ipotesi nulla a livello di significatività 5% e ciò significa che esiste una differenza statisticamente significativa tra le serie di dati. Tuttavia ANOVA non ci consente di determinare quali tra queste si discostano maggiormente le une dalle altre.
Per capire quali medie sono effettivamente diverse tra loro abbiamo utilizzato il *Tukey's Honestly Significant Difference (HSD) test* [3], uno dei tanti test statistici che vengono usati in combinazione con ANOVA e che si basa sui risultati ottenuti dalla sua applicazione. Questo test confronta tutte le possibili coppie di medie e individua dove la differenza, tra due di queste, è maggiore della deviazione standard che ci si aspetterebbe. Utilizza una formula molto simile a quella del t-test ma, a differenza di quest'ultimo, è più adatto per confronti multipli. I risultati che abbiamo ottenuto per i quattro insiemi di esperimenti sono mostrati nelle Figure 5.12, 5.13, 5.14 e 5.15.

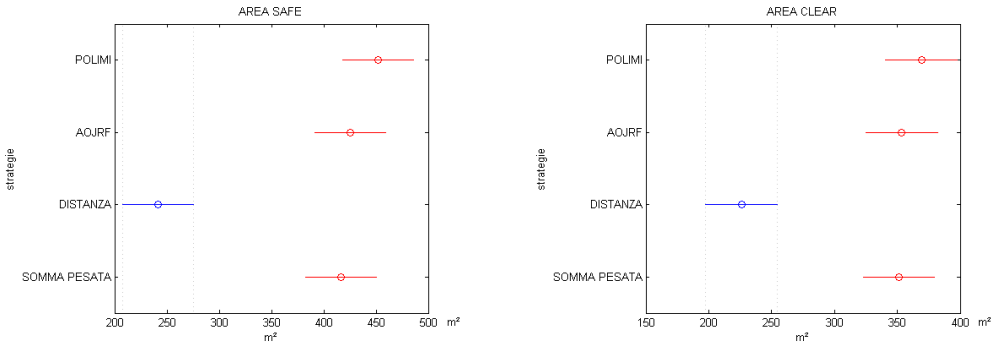


Figura 5.12: Risultati del test HSD per gli esperimenti effettuati con due P2AT nell'hotel

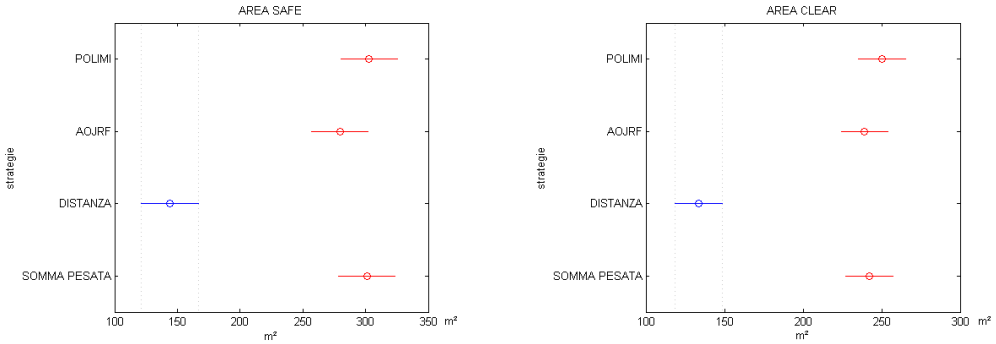


Figura 5.13: Risultati del test HSD per gli esperimenti effettuati con un P2AT nell'hotel

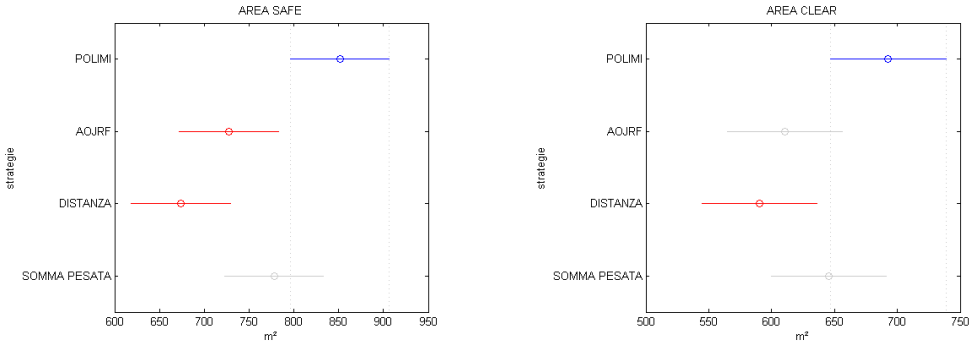


Figura 5.14: Risultati del test HSD per gli esperimenti effettuati con due P2AT nello spazio aperto

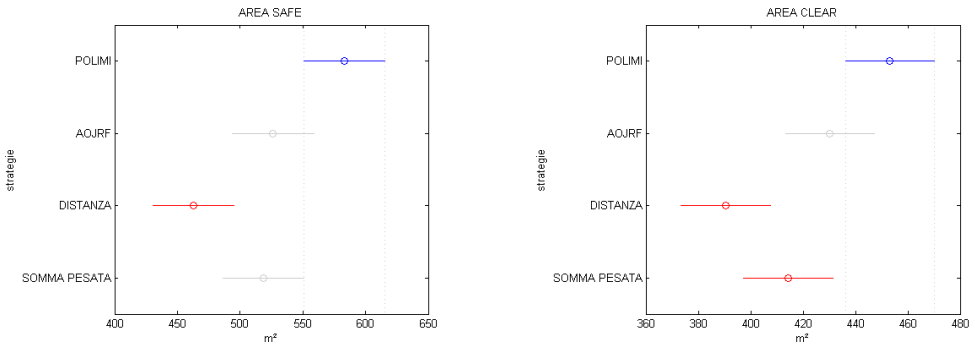


Figura 5.15: Risultati del test HSD per gli esperimenti effettuati con un P2AT nello spazio aperto

Ogni singola immagine mostra, per ogni strategia, il valore medio dell'area, safe o clear a seconda dei casi, esplorata durante gli esperimenti svolti in determinate condizioni, ossia con una certa squadra di agenti in un dato ambiente. Ogni media è rappresentata da un simbolo e da un intervallo di confidenza al $95\%(1-\alpha)$ intorno ad esso. Tra due strategie esiste pertanto una differenza significativa se tali intervalli sono disgiunti, viceversa non sussiste se questi si sovrappongono. Nei grafici in blu è indicato il simbolo associato al metodo di esplorazione che abbiamo selezionato, mentre sono colorati di rosso quelli associati alle strategie con medie significativamente diverse da quello preso in considerazione e di grigio i rimanenti. Ottenendo sulla base dei dati osservati una certa differenza tra le medie campionarie, dire che questa differenza è statisticamente significativa al livello 5% equivale a dire che, accettando H_0 (ossia se le medie fossero uguali), una differenza pari almeno a quella osservata, si osserverebbe meno del 5% delle volte. Detto ciò, dai risultai ottenuti possiamo trarre le seguenti conclusioni:

- Nell'hotel, a conferma di quanto avevamo già dedotto, indipendentemente dal numero di agenti utilizzato, possiamo distinguere due gruppi di strategie tra i quali esiste una differenza statisticamente significativa. Da una parte DISTANZA, che è quella con prestazioni peggiori, e dall'altra POLIMI, AOJRF e SOMMA PESATA, che si sono rivelati metodi di esplorazione paragonabili tra loro in termini di area esplorata.

- All'interno dello spazio aperto, al contrario, non possiamo individuare due gruppi di strategie come nel caso precedente. Anche in questo ambiente è DISTANZA il metodo che esplora meno area, ma non sussiste più una differenza sostanziale tra questa e tutte le altre strategie. Guardando i grafici relativi allo spazio aperto, in cui abbiamo selezionato la media associata a POLIMI, che è la nostra strategia e quindi quella che più ci interessa, possiamo dire che esiste sempre una differenza statisticamente significativa tra questa e DISTANZA. In alcuni casi, però, si nota una differenza anche fra POLIMI e AOJRF e fra POLIMI e SOMMA PESATA, cosa che nell'hotel non è mai accaduta.

Da tutti i dati raccolti e da tutti gli studi effettuati su di essi possiamo affermare di essere soddisfatti dal funzionamento del nostro metodo di esplorazione che ha mostrato prestazioni paragonabili a quelle di una strategia studiata *ad hoc* e che è stata ottimizzata per partecipare alla RoboCup Rescue Virtual Robot Competition.

5.2 Studio dei risultati della seconda versione del software

La seconda versione del software risolve il problema che si viene a creare in uno spazio aperto e che vede i robot bloccati nel centro dell'area relativa ad una frontiera, grazie alla modifica che permette agli agenti di raggiungere, invece del punto centrale di tale area, un punto scelto casualmente al suo interno. Con questo aggiornamento, applicato a tutte le strategie in esame, vogliamo verificare l'impatto della scelta del punto sull'estensione dell'area mappata e sul numero di blocchi dei robot. La nostra speranza è che le loro prestazioni non peggiorino particolarmente e soprattutto di non dover più intervenire sul percorso dei robot.
La Figura 5.16 e le Tabelle 5.11 e 5.12 mostrano i risultati ottenuti con l'applicazione della seconda versione di POLIMI, AOJRF, DISTANZA e SOMMA PESATA in riferimento a cinque esperimenti, le cui posizioni iniziali sono state scelte tra quelle dei dieci effettuati con la prima versione, nello spazio

aperto e con una squadra composta da una ComStation e un P2AT.

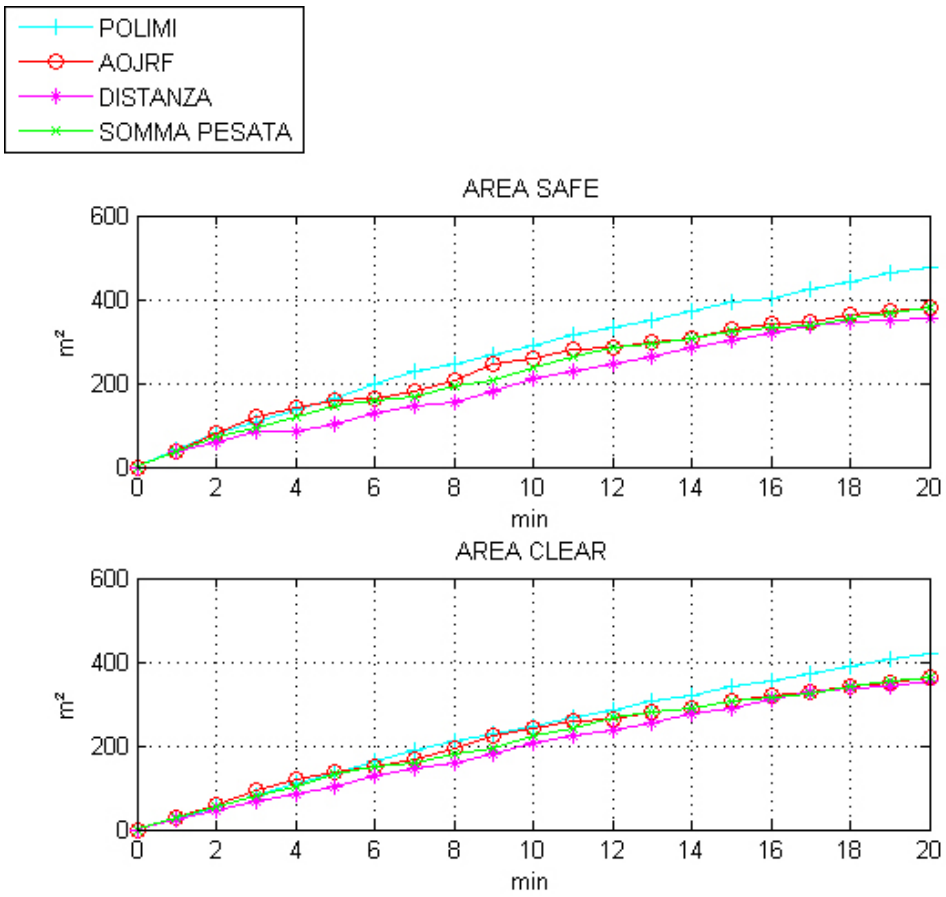


Figura 5.16: Andamento medio delle strategie con un P2AT nello spazio aperto con la seconda versione dei software

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**518.09**	438.85	420.97	284.08
6	**470.99**	311.59	356.48	357.43
7	**490.45**	440.33	326.08	421.00
8	**448.10**	372.73	348.19	344.20
9	458.59	346.30	329.09	**485.23**

Tabella 5.11: Valori dell'area safe ricavati con un P2AT nello spazio aperto con la seconda versione dei software

ESPERIMENTO	POLIMI	AOJRF	DISTANZA	SOMMA PESATA
1	**455.04**	399.89	398.51	286.17
6	**408.24**	332.83	351.02	332.76
7	**417.56**	400.23	331.98	383.56
8	**404.42**	344.75	350.31	360.03
9	404.99	331.04	330.97	**452.77**

Tabella 5.12: Valori dell'area clear ricavati con un P2AT nello spazio aperto con la seconda versione dei software

Per quanto riguarda la quantità di area safe e clear, i valori possono essere confrontati con quelli presenti nelle Tabelle 5.8 e 5.9. I risultati non mostrano miglioramenti nell'estensione dell'area mappata, al contrario il lieve peggioramento è dovuto al fatto che nella prima versione, essendo costretti a muovere il robot verso determinate posizioni per creare ulteriori frontiere e fare in modo che scegliesse un nuovo obiettivo da raggiungere, l'abbiamo portato manualmente a visitare porzioni di area che altrimenti non avrebbe esplorato. In compenso, però, non siamo mai dovuti intervenire nello sbloccare il robot lungo il suo percorso, durante l'esecuzione dei test.
Nella Figura 5.17 presentiamo, a titolo d'esempio, due mappe, risultanti dal-

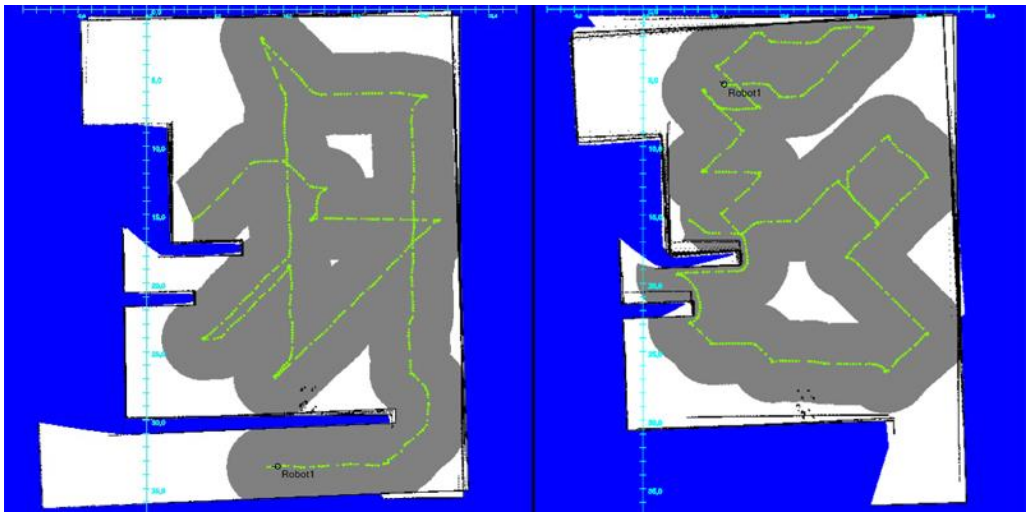


Figura 5.17: Mappe risultanti dall'esperimento 1 con le due versioni di POLIMI

l'esperimento 1 eseguito rispettivamente con la prima e la seconda versione di POLIMI, per mostrare i diversi percorsi del robot, nei due casi. Nel primo vogliamo far notare il tratto orizzontale, che dal centro raggiunge il confine destro della mappa e che rappresenta uno degli interventi da noi effettuati. Nel secondo, invece, il robot ha seguito un percorso meno regolare, cambiando spesso direzione ma non richiedendoci mai di intervenire.

5.3 Studio dei risultati dopo l'introduzione della batteria

Nella Sezione 4.4, sviluppando l'ultima versione del nostro codice siamo già riusciti a dimostrare che, con una strategia basata su MCDM, è possibile introdurre la batteria come nuova informazione da considerare nella valutazione del prossimo punto da raggiungere in due modi diversi, senza dover stravolgere il codice o studiare dall'inizio una nuova formula che la includa. Abbiamo chiamato le due soluzioni, pensate per questo scopo, POLIMI + BATTERIA e POLIMI DOPPI PESI e, per ognuna di queste, abbiamo effettuato cinque esperimenti per confrontarne le prestazioni. Questi test sono stati effettuati con una squadra composta da tre robot, ovvero due P2AT e una ComStation, in entrambi gli ambienti con posizioni iniziali scelte tra quelle già utilizzate in precedenza. In quest'ultima sezione riportiamo i risultati ottenuti mostrando nelle Tabelle 5.13 e 5.14 e nella Figura 5.18 quelli relativi agli esperimenti svolti nell'hotel, e nelle Tabelle 5.15 e 5.16 e nella Figura 5.19 quelli riguardanti gli esperimenti svolti nello spazio aperto.
La quantità di spazio esplorato, sia in termini di area safe che di area clear, non differisce molto nei due casi e, soprattutto dall'andamento medio di queste aree mostrato nei grafici, si nota che le prestazioni sono molto simili.
Qui è stato mostrato un confronto tra i due metodi utilizzati per l'integrazione della batteria nella nostra strategia, mentre, nella prossima sezione, viene descritto un esperimento svolto, oltre che con POLIMI + BATTERIA e con POLIMI DOPPI PESI, anche con la prima versione di POLIMI.

ESPERIMENTO	POLIMI + BATTERIA	POLIMI DOPPI PESI
1	**399.82**	373.25
6	495.58	**526.30**
7	**496.64**	453.44
8	453.72	**511.09**
10	520.25	**534.24**

Tabella 5.13: Valori dell'area safe ricavati con due P2AT nell'hotel

ESPERIMENTO	POLIMI + BATTERIA	POLIMI DOPPI PESI
1	**335.16**	329.69
6	385.73	**448.08**
7	**413.62**	375.96
8	382.06	**393.50**
10	414.32	**428.98**

Tabella 5.14: Valori dell'area clear ricavati con due P2AT nell'hotel

ESPERIMENTO	POLIMI + BATTERIA	POLIMI DOPPI PESI
2	799.81	**862.36**
4	**748.16**	725.49
6	664.34	**753.82**
7	800.45	**877.43**
10	**756.21**	691.49

Tabella 5.15: Valori dell'area safe ricavati con due P2AT nello spazio aperto

ESPERIMENTO	POLIMI + BATTERIA	POLIMI DOPPI PESI
2	724.44	**747.60**
4	**690.83**	656.71
6	635.71	**657.88**
7	729.62	**743.58**
10	**709.06**	617.05

Tabella 5.16: Valori dell'area clear ricavati con due P2AT nello spazio aperto

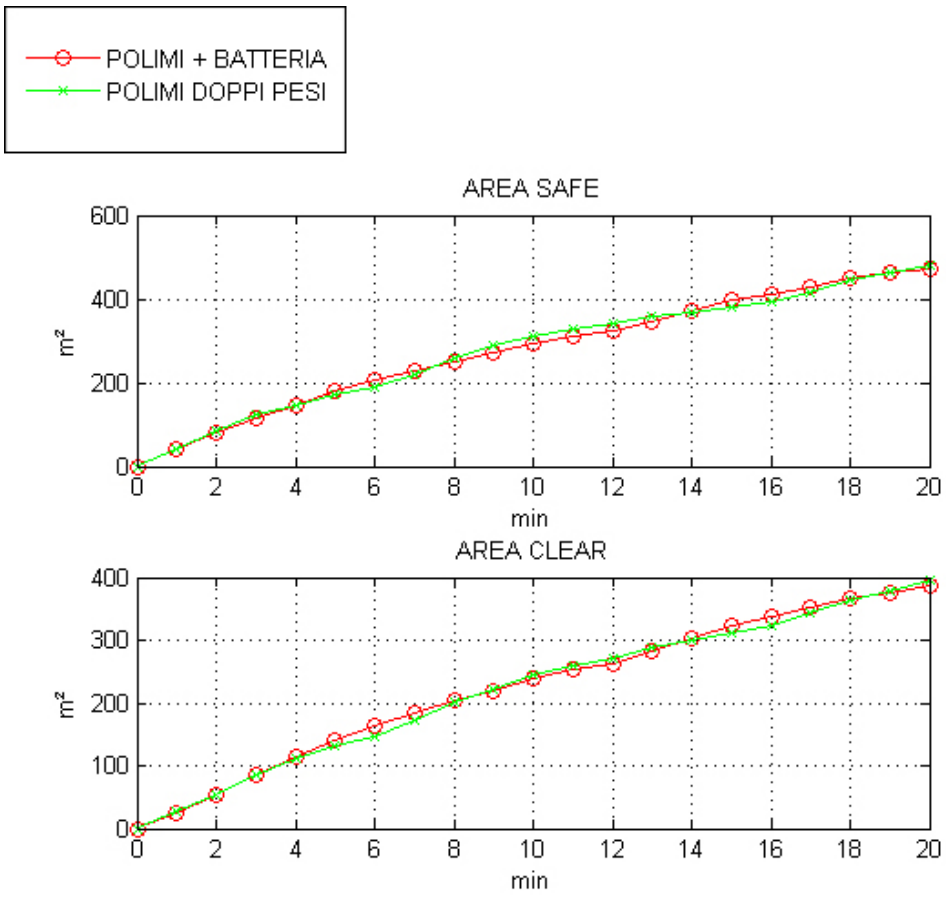


Figura 5.18: Andamento medio delle strategie con due P2AT nell'hotel

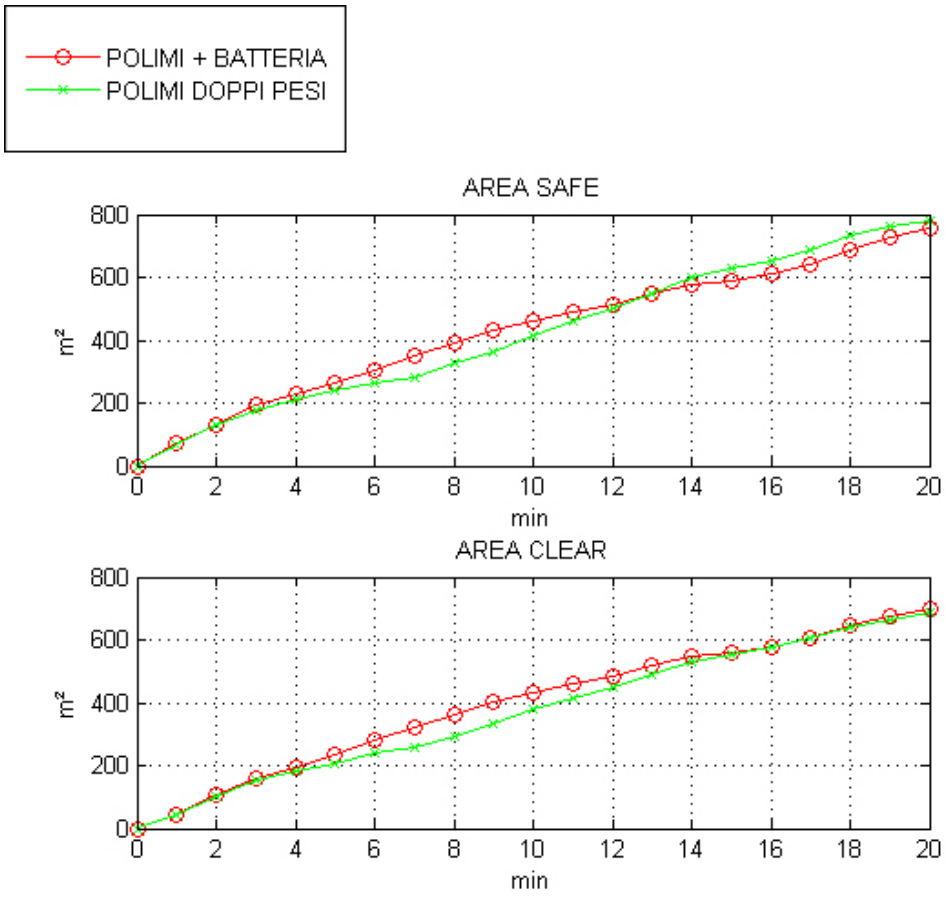


Figura 5.19: Andamento medio delle strategie con due P2AT nello spazio aperto

5.3.1 Esperimento passo-passo

Per studiare le differenze di comportamento che intercorrono tra tre strategie basate su MCDM che differiscono tra loro per il numero e il modo in cui le diverse informazioni vengono considerate nella valutazione dell'utilità di un punto candidato, abbiamo svolto un test specifico, utile per il raggiungimento del nostro scopo.

Partendo dalle posizioni iniziali dell'esperimento 1, effettuato nell'hotel, con una squadra composta da una ComStation e da un P2AT, abbiamo eseguito l'esplorazione con POLIMI, POLIMI + BATTERIA e POLIMI DOPPI PESI. Tra queste, POLIMI e POLIMI + BATTERIA differiscono per il numero di criteri coinvolto, mentre POLIMI e POLIMI DOPPI PESI considerano gli

stessi criteri ma utilizzano diversi insiemi di pesi per attuare diversi comportamenti. I risultati ottenuti sono mostrati in Figura 5.20 per quanto riguarda le mappe create dal robot durante l'esplorazione, relativamente una alla sola area safe e l'altra comprensiva anche dell'area clear e in Tabella 5.17 per quanto riguarda i valori ad esse associati.

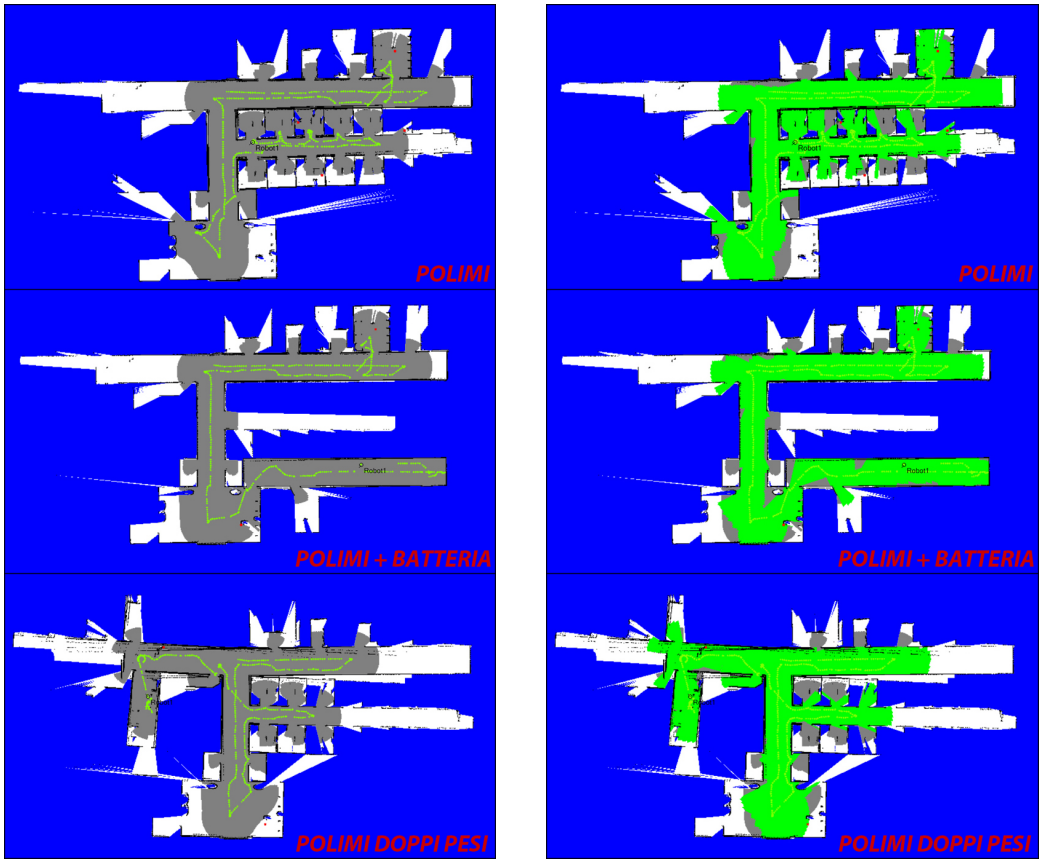


Figura 5.20: Mappe risultanti dall'esperimento passo-passo

AREA	POLIMI	POLIMI + BATTERIA	POLIMI DOPPI PESI
SAFE	**304.99**	285.40	265.23
CLEAR	**271.14**	254.15	237.80

Tabella 5.17: Valori di area risultanti dall'esperimento passo-passo

Tramite questa prova vogliamo valutare passo-passo il comportamento del robot nei tre casi, ossia le sue decisioni, per evidenziare i momenti in cui queste sono significativamente diverse. Da questi risultati si nota subito che POLIMI ha permesso di esplorare più area rispetto alle due strategie che considerano l'utilizzo della batteria, in quanto questa informazione porta il robot ad essere più prudente durante l'esplorazione. Le osservazioni più interessanti, però, si possono fare osservando la Figura 5.21 che mostra i percorsi seguiti dal P2AT nei tre casi, sovrapposti in un'unica immagine che ne permette un facile confronto.

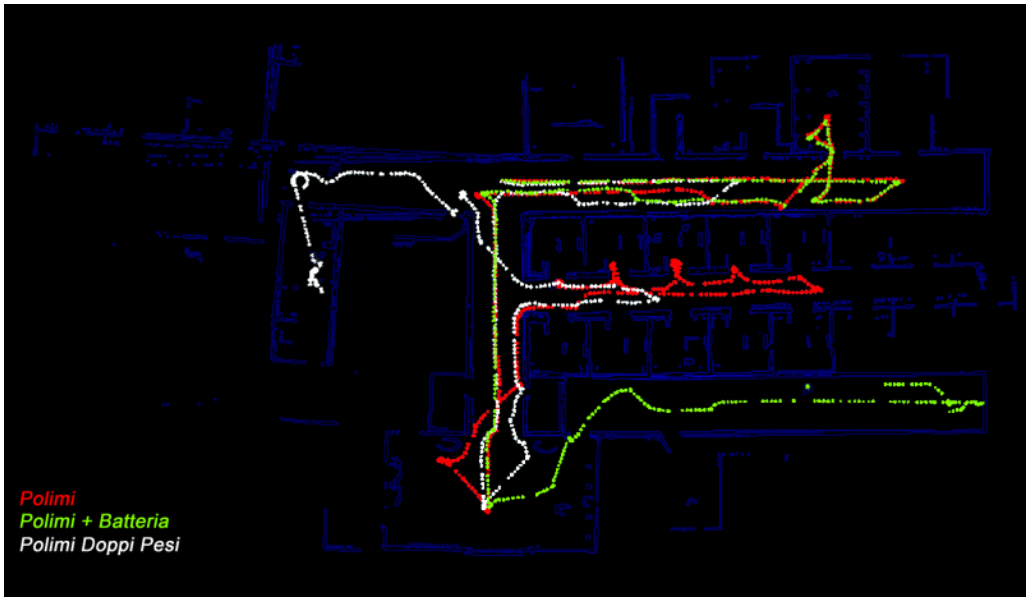


Figura 5.21: Percorsi seguiti dal P2AT utilizzando le tre diverse strategie

Inizialmente il robot, indipendentemente dalla strategia, si è diretto lungo il corridoio in alto a destra, che è quello verso cui è rivolto appena viene inserito nell'ambiente. Lungo il corridoio si può vedere la prima differenza fra i tre metodi di esplorazione. Qui, infatti, mentre con POLIMI e POLIMI + BATTERIA l'agente ha proseguito fino alla fine del corridoio, con POLIMI DOPPI PESI a metà di questo ha scelto di raggiungere una frontiera più lontana ma con un'area stimata maggiore di quella presente davanti a sé (Figura 5.22). Questo a causa del comportamento più aggressivo che gli abbiamo impartito per la prima parte dell'esecuzione della prova, come illu-

strato nella Sezione 4.4.2, assegnando alla strategia determinati pesi mostrati in Tabella 4.7.

(a) POLIMI (b) POLIMI+BATTERIA (c) POLIMI DOPPI PESI

Figura 5.22: Prima differenza nella scelta delle frontiere

Durante l'esplorazione il robot è sempre giunto nell'ingresso, situato in basso al centro sulla mappa, dove, con l'utilizzo di POLIMI + BATTERIA, che fino a quel momento ha avuto un comportamento identico a quello di POLIMI, ha preso una decisione diversa dalle altre e si è portato nel corridoio in basso a destra dove è rimasto fino alla fine dell'esecuzione. POLIMI e POLIMI DOPPI PESI, invece, hanno portato l'agente verso il corridoio centrale parallelo a quest'ultimo (Figura 5.23).

(a) POLIMI (b) POLIMI+BATTERIA (c) POLIMI DOPPI PESI

Figura 5.23: Seconda differenza nella scelta delle frontiere

L'ultima differenza fondamentale vede coinvolte le strategie POLIMI e POLIMI DOPPI PESI che, da qui, fanno in modo che il robot scelga due diversi obiettivi. Con la prima il P2AT ha deciso di continuare a esplorare il corridoio, dove è rimasto fino al termine del test. Con l'altro metodo, invece, ha selezionato una frontiera, come nel caso precedente, più lontana ma con una maggiore area stimata visitabile (Figura 5.24).

(a) POLIMI (b) POLIMI DOPPI PESI

Figura 5.24: Terza differenza nella scelta delle frontiere

Mentre il robot, nel caso di POLIMI DOPPI PESI, si è diretto verso questo punto, sono terminati i primi dieci minuti di esecuzione e, dalle scelte effettuate dopo quest'ultima, si può notare il cambiamento di comportamento da aggressivo a conservativo, che lo ha portato a visitare frontiere più vicine anche se meno interessanti in termini di informazione attesa. Questo fatto è provato dalla forma e dalla lunghezza del percorso che, come si può notare dalla Figura 5.21, è nettamente meno rettilineo e più corto nella seconda parte dell'esperimento.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi abbiamo presentato un approccio basato su MCDM per definire una strategia di esplorazione per robot autonomi impegnati in operazioni di ricerca e soccorso in ambiente urbano. Questo nostro metodo combina diversi criteri in un'unica funzione per determinare la miglior posizione raggiungibile ad ogni passo, permettendoci di considerare la dipendenza tra i criteri nel calcolo dell'utilità. Per raggiungere il nostro scopo ci siamo serviti di un software esistente e completo, sviluppato dai ricercatori delle università di Amsterdam e di Oxford e utilizzato durante la RoboCup Rescue Virtual Robot Competition. Sfruttandolo abbiamo avuto la possibilità di implementare la nostra strategia, sostituendola a quella originale, senza doverci preoccupare di altri aspetti, già sviluppati, quali la localizzazione dei robot, la costruzione della mappa, la pianificazione del percorso, ecc...
Negli esperimenti effettuati abbiamo confrontato la nostra strategia con altri tre metodi di esplorazione. Dai risultati sperimentali abbiamo rilevato che le prestazioni di MCDM, applicato in un contesto di questo tipo, sono paragonabili a quelle di una strategia studiata appositamente per la partecipazione alla RoboCup. Abbiamo, inoltre, dimostrato la flessibilità del nostro approccio considerando un nuovo criterio nel calcolo dell'utilità, senza dover studiare una nuova funzione *ad hoc*. Oltre all'inserimento di un vero e proprio criterio, se la nuova informazione da impiegare nella valutazione di una posizione candidata dipende dalle azioni effettuate in passato dagli agenti,

come è il caso della carica della batteria, MCDM dà la possibilità di utilizzare diversi insiemi di pesi in diverse fasi dell'esplorazione.
Un'ulteriore modifica che abbiamo attuato nei confronti del software originale, motivata dalla risoluzione di un problema riscontrato durante alcuni esperimenti per cui siamo stati costretti ad intervenire manualmente sui movimenti dei robot, ha coinvolto il punto da raggiungere una volta scelto l'obiettivo. Piuttosto di considerare il centro dell'area presente oltre la frontiera scelta, abbiamo optato per un punto casuale al suo interno. Grazie a questa semplice soluzione siamo riusciti a rendere i movimenti dei robot completamente autonomi per tutta la durata dell'esplorazione.
Possibili sviluppi futuri potrebbero migliorare il sistema, a partire dall'algoritmo per il Path Planning che è computazionalmente costoso in quanto calcola il percorso analizzando un'immagine, raffigurante la mappa, e considerando ogni singolo punto che il robot deve attraversare. Una soluzione migliore potrebbe essere quella di utilizzare un grafo di raggiungibilità, facendo in modo di restituire un percorso che comprenda solamente alcuni punti di via. Con un algoritmo di Path Planning più leggero, inoltre, sarebbe possibile calcolare fin dal principio tutti i valori di utilità utilizzando la distanza reale, invece di farlo in due sessioni separate, prima con la distanza euclidea e, successivamente, ma solo per quelli più elevati, con quella reale.
Quando un agente sceglie il suo prossimo obiettivo, per come è stato progettato il codice, considera anche la presenza di altri eventuali componenti della squadra, nel senso che calcola anche per questi la miglior frontiera da raggiungere e utilizza tale informazione, che non condivide con nessuno, per fare in modo di non scegliere frontiere che sarebbero migliori per altri agenti. Questa scelta, però, potrebbe non essere la migliore e per verificare la sua ragionevolezza la si potrebbe confrontare con i due casi estremi, uno in cui ogni agente non considera gli altri nel prendere la decisione e l'altro in cui un robot, ad esempio la ComStation, calcola la miglior frontiera per ogni agente assegnandoli tutti nello stesso istante.
Un'ultima osservazione riguarda la definizione delle frontiere come il confine tra l'area safe e l'area clear, che spinge il robot a visitare zone già "viste" tramite il sensore di distanza con la portata più ampia. Una modifica in

questo senso potrebbe consistere nel considerare come frontiera il limite tra l'area free e l'ambiente inesplorato al di là di essa.
Per quanto riguarda MCDM, un miglioramento che si potrebbe apportare riguarda la definizione dei pesi. In questo lavoro li abbiamo scelti manualmente secondo le nostre esigenze e facendo alcuni esperimenti per verificarne la correttezza. In futuro si potrebbero definire degli algoritmi che permettano di determinarli automaticamente in base all'obiettivo da perseguire per fare in modo di ottimizzare la funzione di utilità.

Bibliografia

[1] F. Amigoni, V. Caglioti, and U. Galtarossa. A mobile robot mapping system with an information-based exploration strategy. In *Proceedings ICINCO*, pages 71–78, 2004.

[2] S. Balakirsky, S. Carpin, A. Kleiner, M. Lewis, A. Visser, J. Wang, and V. A. Ziparo. Towards heterogeneous robot teams for disaster mitigation: Results and Performance Metrics from RoboCup Rescue. *Journal of Field Robotics*, 24(11–12):943–967, 2007.

[3] J. Barnette and J. McLean. The Tukey Honestly Significant Difference Procedure and Its Control of the Type I Error-Rate. In *Annual Meeting of the Mid-South Educational Research Association*, 1998.

[4] N. Basilico and F. Amigoni. Exploration Strategies based on multi-Criteria Decision Making for an Autonomous Mobile Robot. In *Proceedings of the European Conference on Mobile Robots*, pages 259–264, 2009.

[5] W. Burgard, M. Moors, and F. Schneider. Collaborative Exploration of Unknown Environments with Teams of Mobile Robots. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents*, pages 52–70, London, UK, 2002.

[6] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated Multi-Robot Exploration. *IEEE Transactions on Robotics*, 21:376–386, 2005.

[7] D. Calisi, A. Farinelli, L. Iocchi, and D. Nardi. Multi-objective Exploration and Search for Autonomous Rescue Robots. *Journal of Field Robotics, Special Issue on Quantitative Performance Evaluation of Robotic and Intelligent Systems*, 24:763–777, August - September 2007.

[8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006.

[9] R. A. Fisher. Applications of Student's distribution. *Metron*, 5:90–104, 1925.

[10] H. Gonzáles-Baños and J. C. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research*, 21(10–11):829–848, 2002.

[11] M. Grabisch and C. Labreuche. A decade of application of the Choquet and Sugeno integrals in multi-criteria decision aid. *4OR: A Quarterly Journal of Operations Research*, 6(1):1–44, 2008.

[12] H. Kitano and S. Tadokoro. RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine*, 22(1):39–52, 2001.

[13] A. Kleiner and V.A. Ziparo. RoboCupRescue - Simulation League Team RescueRobots Freiburg (Germany). In *RoboCup 2006 (CDROM Proceedings), Team Description Paper, Rescue Simulation League*, 2006.

[14] T. Murata. Petri nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[15] M. Pfingsthorn. *RoboCup Rescue Virtual Robots: Wireless Simulation Server Documentation*, October 2008.

[16] P. Scerri, Y. Xu, E. Liao, J. Lai, M. Lewis, and K. Sycara. Coordinating very large groups of wide area search munitions. In D. Grundel, R. Murphey, and P. Pandalos, editors, *Recent Developments in Cooperative Control and Optimization*, pages 451–480. World Scientific Publishing, Singapore, 2004.

[17] H. Scheffé. *The Analysis of Variance*. Wiley-Interscience, 1999.

[18] B. Slamet and M. Pfingsthorn. ManifoldSLAM: a Multi-Agent Simultaneous Localization and Mapping System for the RoboCup Rescue Virtual Robots Competition. Master's thesis, Universiteit van Amsterdam, December 2006.

[19] C. Stachniss and W. Burgard. Exploring Unknown Environments with Mobile Robots using Coverage Maps. In *Proceedings IJCAI*, pages 1127–1134, 2003.

[20] B. Tovar, L. Muñoz-Gómez, R. Murrieta-Cid, M. Alencastre-Miranda, R. Monroy, and S. Hutchinson. Planning exploration strategies for simultaneous localization and mapping. *Robotics and Autonomous Systems*, 54(4):314–331, 2006.

[21] P. Velagapudi, J. Kwak, P. Scerri, M. Lewis, and K. Sycara. Robocup Rescue - Virtual Robots Team STEEL (USA) - MrCS - The Multirobot Control System. In *Proceedings CD of the 12th RoboCup Symposium*, 2008.

[22] A. Visser, T. Schmits, S. Roebert, and J. de Hoog. Amsterdam Oxford Joint Rescue Forces - Team Description Paper - Virtual Robot competition - Rescue Simulation League - RoboCup 2008. In *Proceedings CD of the 12th RoboCup Symposium*, Suzhou, China, 2008.

[23] A. Visser and B. A. Slamet. Including communication success in the estimation of information gain for multi-robot exploration. In *International Workshop on Wireless Multihop Communications in Networked Robotics*, Berlin, 2008.

[24] J. Wang and S. Balakirsky. *USARSim V3.1.3 - A Game-based Simulation of mobile robots*. http://sourceforge.net/projects/usarsim/, 2009.

[25] B. Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*, pages 146–151, 1997.

[26] B. Yamauchi. Frontier-Based Exploration Using Multiple Robots. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 47–53, 1998.

[27] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H. Costelha. Petri Net Plans. In *Proceedings of Fourth International Workshop on Modelling of Objects, Components, and Agents (MOCA)*, pages 267–290, Turku, Finland, 2006.

[28] V. A. Ziparo, A. Kleiner, B. Nebel, and D. Nardi. RFID-Based Exploration for Large Robot Teams. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4606–4613, Rome, Italy, 2007.

Appendice A

Manuale utente

Di seguito vengono illustrate le procedure per effettuare l'installazione del nostro progetto e per imparare a configurarlo in modo da poterlo utilizzare nel modo corretto.
È necessario sottolineare che l'esecuzione del programma richiede ingenti risorse computazionali e che noi, infatti, siamo riusciti ad effettuare esperimenti utilizzando squadre di robot composte da non più di due agenti e una ComStation, sfruttando un computer con quattro processori e sei Gigabyte di RAM. Detto questo, per procedere con l'installazione è necessario possedere un computer montante Windows XP o superiore come sistema operativo e, siccome USARSim si basa sul motore grafico di Unreal Tournament 2004, ed essendo questo software a pagamento, acquistare una licenza di utilizzo a parte.

A.1 Installazione dell'ambiente di lavoro

Una volta installato il software proprietario, seguendo pochi passi fondamentali sarà possibile disporre di un sistema funzionante e ottimizzato per svolgere il tipo di lavoro richiesto e quindi riuscire a sfruttare al meglio le potenzialità del nostro progetto.
Attenendosi alle istruzioni che compaiono sullo schermo, si può procedere con l'installazione del materiale, presente nella cartella `Installazione` del

DVD, nel modo seguente, ricordandosi di sovrascrivere ogni volta che viene richiesto:

- Installare la patch `UT2004-WinPatch3369.exe`[1], in quanto USARSim è ottimizzato per questa versione di Unreal Tournament 2004.
- Procedere all'installazione dell'ambiente di lavoro, estraendo il file `USARSimFull_3.31.zip`[2] direttamente nella cartella di installazione di Unreal Tournament 2004, che di default è `C:\UT2004`. Da ora in poi, quando parleremo della cartella `C:\UT2004`, ci riferiremo a quella in cui è stato installato Unreal Tournament 2004, se in fase di installazione è stato scelto un percorso diverso da quello di default si dovrà fare riferimento a tale percorso.
- Installare le varie mappe contenute nella cartella `Maps`[2] del DVD allegato, estraendole in `C:\UT2004`.
- Estrarre il file `UT2004.zip`[2] in `C:\UT2004`.
- Completare l'installazione compilando il programma tramite l'esecuzione del file `make.bat` presente nella cartella `C:\UT2004\System`.

Una volta eseguiti tutti i punti precedenti ed effettuata la compilazione si avrà USARSim installato sul proprio sistema e perfettamente funzionante. Per la corretta esecuzione del software da noi creato è necessario installare anche `ImageSrv_3.1.zip`[2] e `WSS-0.6.1-win32.zip`[3], estraendoli dove si vuole. Image Server permette di visualizzare a schermo l'ambiente di lavoro del robot e il robot stesso che si muove al suo interno. WSS, invece, ha lo scopo di simulare il traffico Wireless/LAN scambiato tra i robot che comunicano all'interno del mondo virtuale.
Infine estrarre dove si vuole il file `POLIMI V.X.zip`, contenente il software da noi sviluppato usando il linguaggio di programmazione Visual Basic .NET.
A questo punto tutto è pronto per poter effettivamente utilizzare il nostro programma.

[1] http://data.unrealtournament.com/UT2004-WinPatch3369.exe
[2] http://sourceforge.net/projects/usarsim/files/
[3] http://robotics.jacobs-university.de/VirtualRobots/WSS.html

A.2 POLIMI: Configurazione e modalità di funzionamento

L'utilizzo di POLIMI prevede che, come prima cosa, venga creata una squadra di agenti da disporre all'interno della mappa che si vuole esplorare. Per creare questa squadra, che deve essere composta da una ComStation e da uno o più robot, bisogna procedere nel seguente modo:

- Eseguire UsarCommander cliccando sul collegamento `Shortcut to UsarCommander` presente nella cartella POLIMI. Comparirà una schermata blu, su cui successivamente verrà mostrata la mappa inizialmente vuota, con un riquadro nero sulla destra. All'interno di quest'ultimo sono posizionati tutti i pulsanti necessari per configurare il team di robot, come mostrato nella Figura A.1.

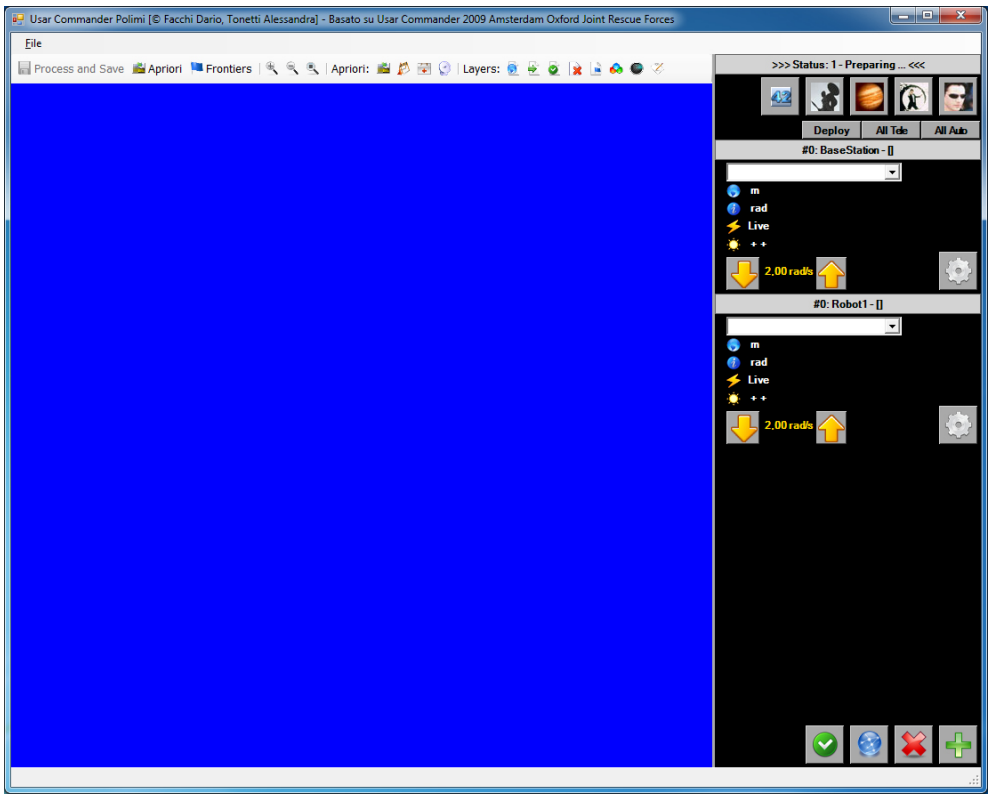


Figura A.1: Schermata iniziale di UsarCommander

- Aggiungere un robot cliccando su  e continuare finchè non si è raggiunto il numero di robot desiderato per la propria squadra. I robot inseriti devono essere almeno due, tra cui una ComStation.

- Configurare ciascun robot. Per fare ciò, per ogni agente, bisogna cliccare su  . Così facendo comparirà una schermata, come quella mostrata in Figura A.2, in cui è possibile inserire le impostazioni di configurazione desiderate, ricordandosi di impostare sempre come `Driver Type` l'opzione `Live Mode` per potersi connettere a USARSim.

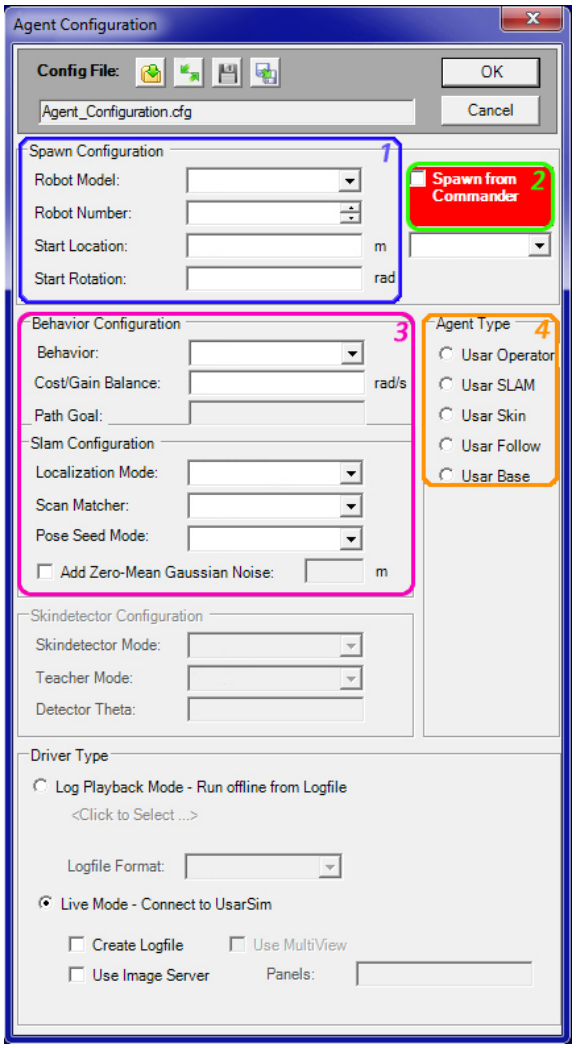


Figura A.2: Schermata di configurazione di un agente

Qui è necessario impostare il modello del robot, il numero ad esso associato (che deve essere diverso per ogni componente della squadra), quali sono la sua posizione di partenza, espressa tramite una terna di coordinate (x, y, z) in base al sistema di riferimento assoluto dell'ambiente che si vuole esplorare, la sua rotazione iniziale definita in radianti (1) e spuntare la casella `Spawn from Commander` (2), solo se si sta considerando una ComStation. Bisogna, inoltre, configurare la strategia da utilizzare, il `Localization Mode`, lo `Scan Matcher` e il `Pose Seed Mode` (3) e selezionare il tipo di agente che si vuole utilizzare (4). Dopo aver impostato posizione, rotazione, strategia di movimento e tutti gli altri parametri di ogni singolo robot, prima di passare al prossimo, si deve salvare la configurazione in un file `Agent_Configuration.cfg` nella cartella `POLIMI\Code\Usar\UsarClient\bin\Release`.

- Cliccare su  per impostare i parametri della rete. Dato che è possibile distribuire il lavoro su più computer, nella finestra che compare, mostrata nella Figura A.3, si devono inserire gli Indirizzi IP delle macchine su cui vengono eseguiti i vari componenti. Se si lavora su un solo pc bisogna inserire in tutti i campi l'indirizzo Localhost (127.0.0.1).

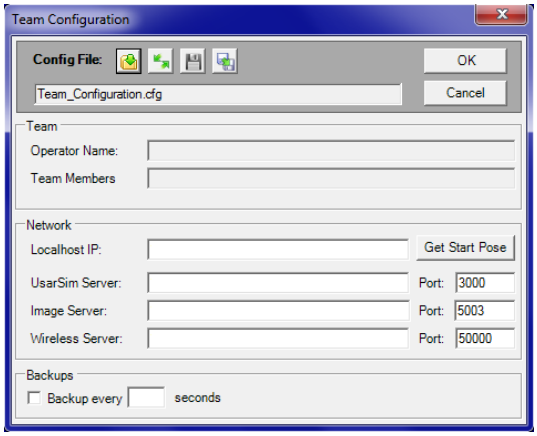


Figura A.3: Schermata di configurazione della squadra di agenti

L'utilizzo di WSS per la gestione della comunicazione prevede che la porta del Wireless Server venga impostata a 50000 mentre le altre devono essere lasciate inalterate. Inoltre, è anche possibile fare

in modo che, con regolarità, il sistema restituisca delle informazioni utili, riguardanti l'esplorazione eseguita dai robot, settando un intervallo di backup in secondi. Questi dati vengono salvati nella cartella `POLIMI\POLIMI_REPORTS` che si creerà durante l'esecuzione del programma. Anche in questo caso si deve salvare il tutto in un file `Team_Configuration.cfg` nella cartella `POLIMI\Code\Usar\UsarClient\bin\Release`.

- Cliccare su  per passare alla fase successiva.

Prima di proseguire bisogna avviare WSS e Image Server. Per quanto riguarda WSS, la cui schermata iniziale è mostrata in Figura A.4, basta solo eseguire il file `WSS.exe` presente all'interno della cartella del programma e cliccare su `Start`.

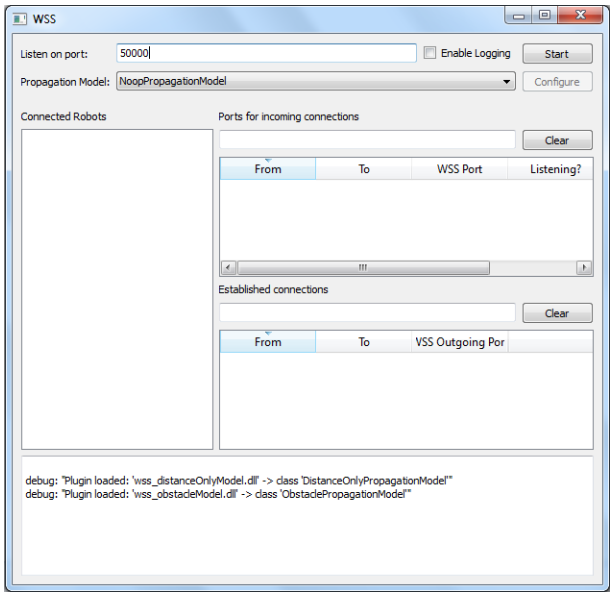


Figura A.4: WSS

Per quanto riguarda Image Server, la cui schermata iniziale è mostrata nella Figura A.5, invece, bisogna seguire quattro semplici passi:

- Eseguire il file `ImageSrv.exe` presente nella cartella `Release` all'interno della directory del programma.

- Spuntare `UT Client Mode`, in `UT Map` inserire il nome della mappa che si vuole utilizzare e premere `Start`.

- Selezionare `UT2004.exe`, invece di `UT2003.exe`, nella finestra che appare e cliccare su `Apri` per avviare l'esecuzione del programma.

- Premere `Show UT` in modo da poter visualizzare la finestra con la mappa scelta, all'interno della quale è possibile spostarsi utilizzando mouse e tastiera.

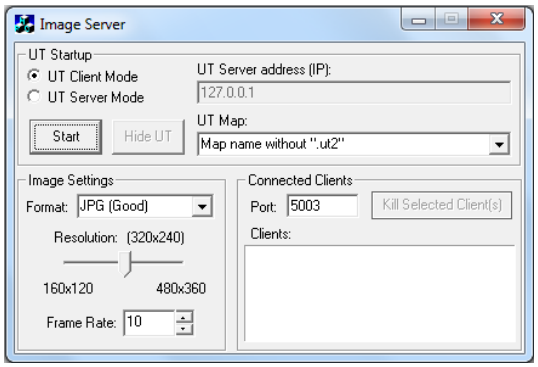


Figura A.5: Image Server

Ora si può procedere con l'inserimento dei robot all'interno della mappa, nel seguente modo:

- In UsarCommander cliccare su 

.

- Depositare la ComStation cliccando su  .

- Una volta che la ComStation è comparsa nella mappa si può procedere all'inserimento degli altri robot. Per ognuno di essi bisogna aprire un prompt dei comandi e cliccare su  . Dal prompt bisogna portarsi nella cartella `POLIMI\Code\Usar\UsarClient\bin\Release`, dove si trova il file `UsarClient.exe`, digitare ed eseguire il comando `UsarClient.exe -n Robot_Name -ac Agent_Configuration.cfg -tc Team_Configuration.cfg` per far partire il robot.

Gli agenti inizieranno a muoversi non appena toccheranno terra e si creerà la mappa.
Da UsarCommander è possibile abilitare o disabilitare differenti livelli della mappa, cerchiati in rosso nella Figura A.6, ognuno corrispondente ad un diverso elemento (area free, area safe, area clear, vittime, ostacoli, ecc...). In ogni momento, inoltre, è possibile modificare la strategia di movimento del robot, semplicemente scegliendone una tra quelle presenti nel menu a tendina che si trova all'interno del riquadro nero di UsarCommander. In modalità Tele Operation è possibile comandare il robot manualmente, mandandolo dove si vuole, utilizzando gli appositi pulsanti, che appaiono cliccando su  , presenti in UsarCommander e mostrati in Figura A.6.

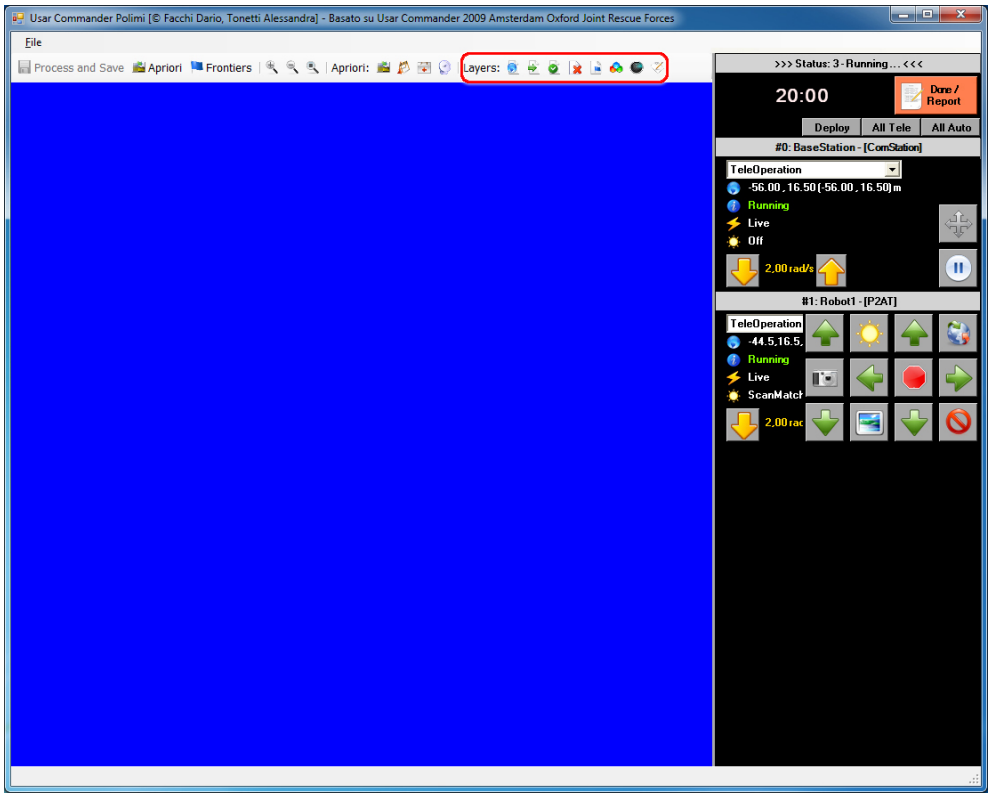


Figura A.6: UsarCommander in modalità Tele Operation

Alla fine dell'esecuzione basta cliccare su 

e successivamente su 

per terminare l'esperimento.

Appendice B

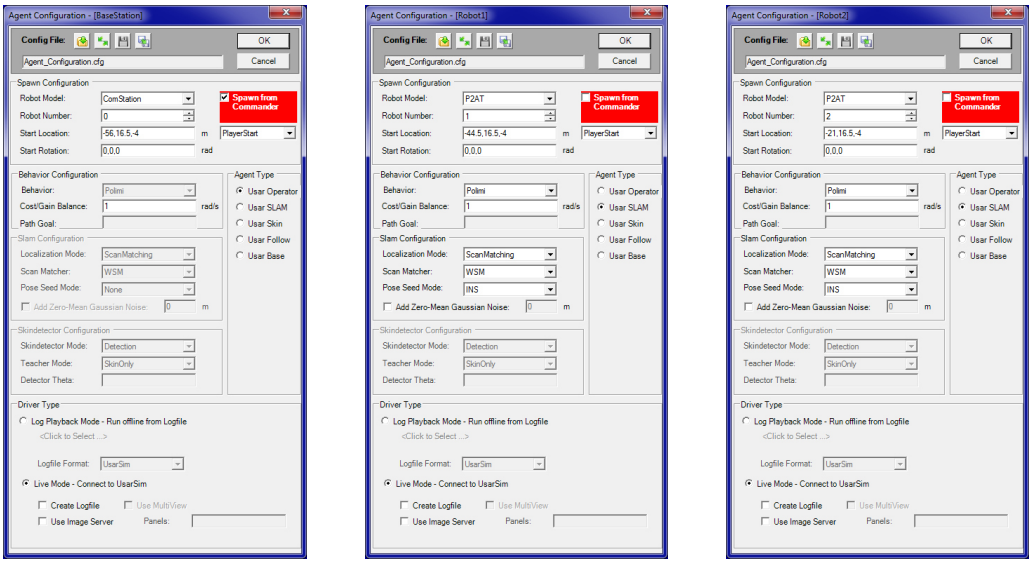
Esempio di utilizzo del sistema

In questa sede vogliamo mostrare tutti i passaggi necessari per dare a chiunque la possibilità di riprodurre uno dei nostri test. Farlo non è complicato, è solamente necessario seguire le istruzioni riportate nell'Appendice A e utilizzare le nostre stesse impostazioni per la configurazione degli agenti.

Riportiamo, quindi, un esempio di utilizzo del sistema fatto impiegando:

- POLIMI come strategia di esplorazione.
- Una squadra di robot composta da tre agenti, una ComStation e due P2AT.
- L'hotel come ambiente virtuale da esplorare.
- Le posizioni iniziali (x, y, z) dell'esperimento 1 relativo all'ambiente considerato.
  - ComStation: -56, 16.5, -4.
  - Robot1: -44.5, 16.5, -4.
  - Robot2: -21, 16.5, -4.

La Figura B.1 mostra quali sono le opzioni da settare per ogni componente della squadra dopo averlo inserito nella schermata di UsarCommander.



(a) ComStation (b) Robot1 (c) Robot2

Figura B.1: Schermate di configurazione degli agenti utilizzati

Per quanto riguarda la ComStation è necessario, oltre ad impostare modello, numero, posizione e rotazione[1] iniziali, solamente spuntare `Spawn from Commander`, in modo che questa venga generata da UsarCommander, e scegliere come `Agent Type` l'opzione `Usar Operator` creata appositamente per gestire questo tipo di agente.

Nelle impostazioni relative agli altri due robot non bisogna spuntare `Spawn from Commander`, in modo che vengano generati da uno UsarClient, si deve scegliere `Usar SLAM`, il tipo più adatto per l'esplorazione autonoma, come `Agent Type` e, naturalmente, è necessario inserire modello, numero, posizione e rotazione di partenza dell'agente. Dalla figura si può notare che abbiamo impostato POLIMI, ovviamente, come strategia di esplorazione e `ScanMatching` come metodo di localizzazione. Questo, infatti, permette al robot di localizzare se stesso nell'ambiente basandosi sulle osservazioni della mappa, ed è meno sensibile agli errori rispetto a `DeadReckoning`, che utilizza le posizioni precedenti del robot per definire quella attuale. In questo lavoro, inoltre, consideriamo come `Scan Matcher` il `Weighted Scan Matcher (WSM)`

[1] In tutti gli esperimenti abbiamo sempre mantenuto $(0, 0, 0)$ come valori iniziali di rotazione dei robot.

e come `Pose Seed Mode` il sensore `Inertial Navigation System (INS)`, che è migliore degli altri disponibili.
Tra i tipi di robot messi a disposizione alcuni non vengono considerati in questo lavoro, ossia `Usar Follow`, `Usar Base` e `Usar Skin`, quest'ultimo permettendoci di non preoccuparci della parte relativa alla `Skindetector Configuration`, strettamente associata ad esso.
Dopo aver salvato ogni configurazione nel file `Agent_Configuration.cfg`, abbiamo impostato i parametri della rete come mostrato in figurename B.2, salvando anch'essi in un file `Team_Configuration.cfg`.

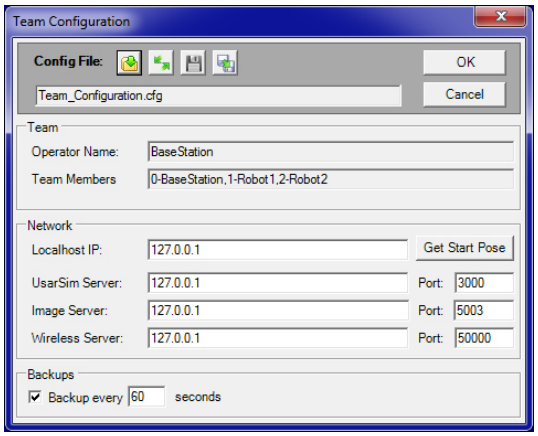


Figura B.2: Schermata di configurazione della nostra squadra di agenti

Una volta avviati WSS e Image Server con la mappa `DM-compWorldDay2_250`, è possibile iniziare la simulazione depositando al suo interno prima la Com-Station e successivamente i due P2AT. Al termine dei 20 minuti di test è possibile visualizzare nella finestra principale di UsarCommander la rappresentazione dell'ambiente creata dai robot durante l'esplorazione. Nelle Figure B.3 e B.4 è mostrato il risultato finale che abbiamo ottenuto con questo esperimento. Come già spiegato, tramite UsarCommander è possibile abilitare o disabilitare diversi livelli della mappa. Noi, nella Figura B.3 abbiamo visualizzato l'area free, l'area safe, le vittime, gli ostacoli e il percorso dei robot, mentre nella Figura B.4 a questi abbiamo aggiunto anche l'area clear.

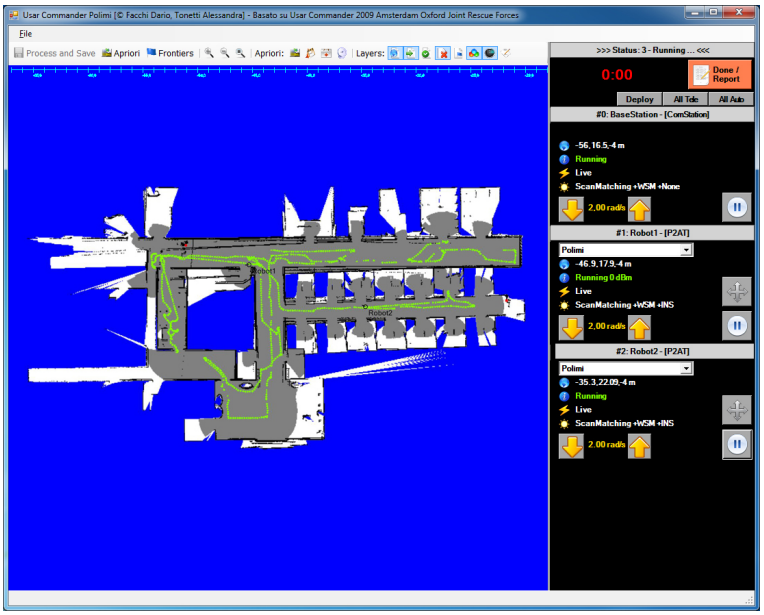


Figura B.3: Mappa risultante

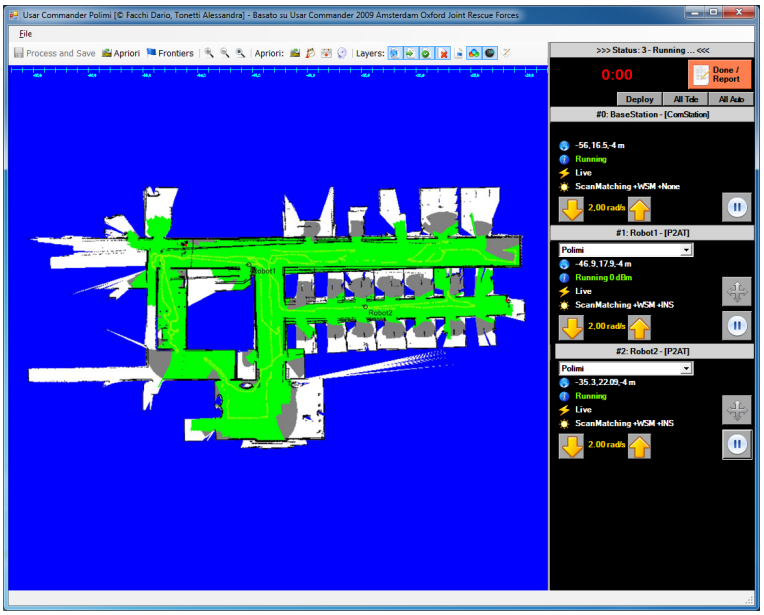


Figura B.4: Mappa risultante comprensiva dell'area clear

Nel caso in cui si scelga di fare un test sotto altre condizioni (diversa strategia, mappa, numero di agenti, ecc...) le impostazioni appena descritte devono, ovviamente, variare di conseguenza.

Appendice C

Posizioni iniziali utilizzate negli esperimenti

Per fare in modo che tutti possano ripetere gli esperimenti da noi effettuati elenchiamo di seguito tutte le posizioni iniziali che abbiamo considerato, relativamente ai componenti delle nostre squadre di robot. Per ogni insieme di coordinate riportiamo anche la mappa ad esso relativa, in cui sono segnalate le ubicazioni della ComStation, in giallo, del Robot1, in rosso, e del Robot2, in verde. In grassetto, inoltre, indichiamo i robot utilizzati negli esperimenti che coinvolgono squadre composte solo da due elementi.
Le posizioni iniziali relative agli esperimenti effettuati nella mappa, chiamata `DM-compWorldDay2_250`, raffigurante un piano di un hotel sono le seguenti:

Esperimento 1
ComStation: -56, 16.5, -4
Robot1: -44.5, 16.5, -4
Robot2: -21, 16.5, -4

Esperimento 2
ComStation: -44.5, 15.5, -4
Robot1: -21, 16.5, -4
Robot2: -56, 16.5, -4

Esperimento 3

ComStation: -21, 16.5, -4

Robot1: -44.5, 16.5, -4

Robot2: -56, 16.5, -4

Esperimento 4

ComStation: -31, 13.5, -4

Robot1: -36.4, 30.1, -4

Robot2: -56.3, 27.4, -4

Esperimento 5

ComStation: -55.1, 20, -4

Robot1: -23.7, 23.3, -4

Robot2: -62.5, 34, -4

Esperimento 6

ComStation: -58.7, 17.3, -4

Robot1: -46.4, 33.6, -4

Robot2: -62, 25.3, -4

Esperimento 7

ComStation: -54, 14, -4

Robot1: -51.2, 19.5, -4

Robot2: -59.4, 10.8, -4

Esperimento 8

ComStation: -29.5, 11.5, -4

Robot1: -64.2, 20.8, -4

Robot2: -40.7, 22.3, -4

Esperimento 9

ComStation: -26.9, 33.2, -4

Robot1: -54.2, 26, -4

Robot2: -35.8, 11, -4

Esperimento 10

ComStation: -51.8, 34.7, -4

Robot1: -28, 28, -4

Robot2: -43, 27.4, -4

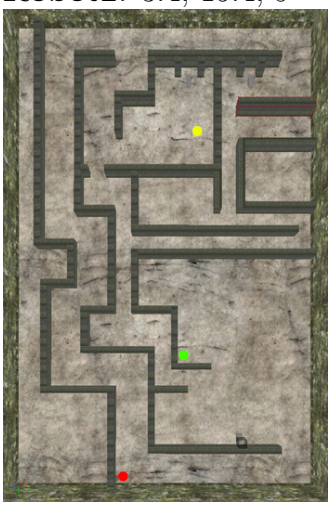
Per quanto riguarda gli esperimenti svolti nella mappa, chiamata `DM-VMAC1`, raffigurante uno spazio aperto, invece, le posizioni iniziali sono le seguenti:

Esperimento 1

ComStation: 5, -19, 9

Robot1: -6.1, 34.7, 9

Robot2: 3.4, 15.4, 9

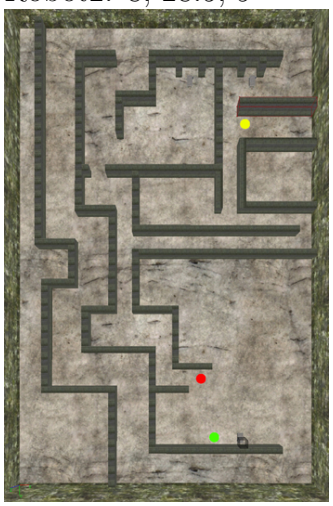


Esperimento 2

ComStation: 15, -21, 9

Robot1: 4.8, 19.4, 9

Robot2: 8, 28.5, 9

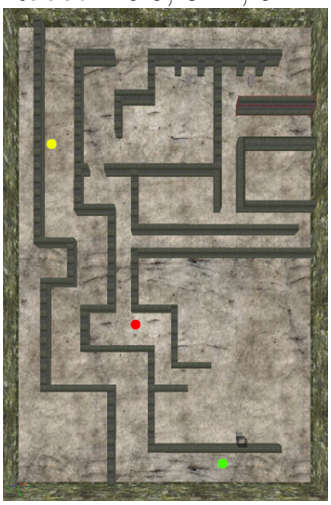


Esperimento 3

ComStation: -18.5, -17.7, 9

Robot1: -5.3, 10.3, 9

Robot2: 6.5, 32.4, 9

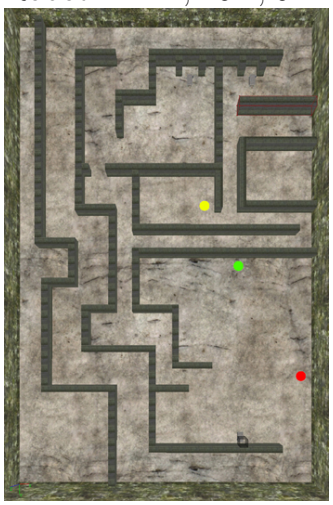


Esperimento 4

ComStation: 7, -7, 9

Robot1: 11.5, 1.9, 9

Robot2: 21.2, 15.1, 9

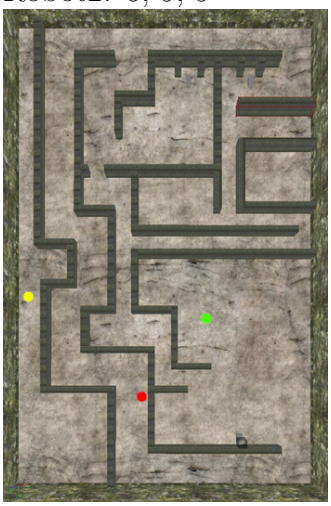


Esperimento 5

ComStation: -19, 8.4, 9

Robot1: -3.2, 20, 9

Robot2: 6, 9, 9

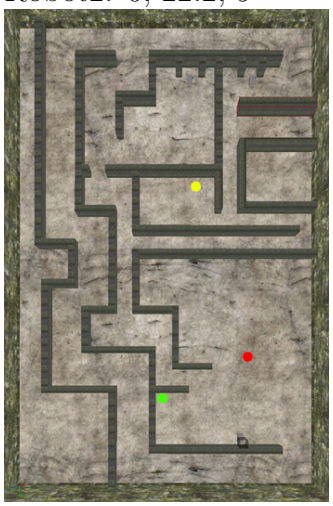


Esperimento 6

ComStation: 5.2, -12.3, 9

Robot1: 13.5, 14.3, 9

Robot2: 0, 22.2, 9

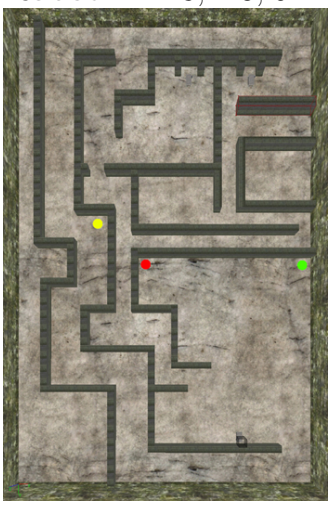


Esperimento 7

ComStation: -10.4, -4.4, 9

Robot1: -2.7, 1.3, 9

Robot2: 21.8, 1.3, 9

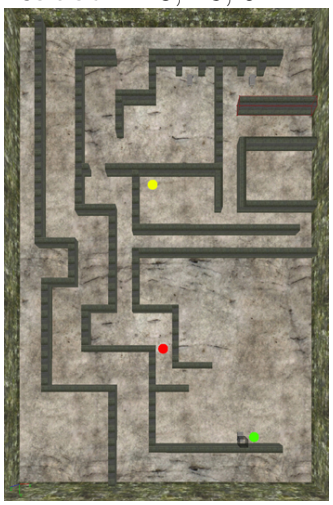


Esperimento 8

ComStation: -2, -11.5, 9

Robot1: 0, 14, 9

Robot2: 13, 28, 9

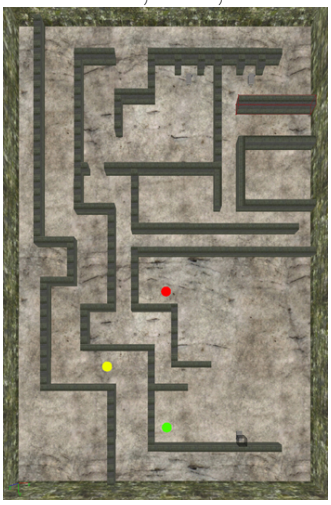


Esperimento 9

ComStation: -9, 18, 9

Robot1: 0, 5.2, 9

Robot2: 0, 27.3, 9



Esperimento 10

ComStation: -5, -3.2, 9

Robot1: 21.4, 26, 9

Robot2: 7, 7.7, 9