Assegnamento Nov-Dic 2022
Corso di Programmazione di Sistemi Embedded

Sommario
In questo assegnamento vi occuperete del controllo di una flotta di robot mobili che si muove sulla superficie di Marte per recuperare dei campioni.
L'assegnamento si compone di tre passi:

1. Sviluppo del sistema di pianificazione di un robot per portare il robot dalla posizione iniziale alla posizione obiettivo senza collidere con gli ostacoli.
2. Estensione del problema a più robot in movimento sulla superficie. I robot dovranno evitare gli ostacoli e gli altri robot in movimento. Nuovi obiettivi saranno aggiunti man mano che i robot raggiungono quelli indicati precedentemente.
3. Il sistema sviluppato verrà infine collegato a una simulazione tramite ROS per fornire una rappresentazione grafica dei percorsi calcolati.

Keywords
Pianificazione del moto – Campi di potenziale artificiale — C++ — Concorrenza – ROS

Indice

1 Pianificazione del moto **1**
1.1 Concetti di Base 2
2 Metodo dei campi di potenziale **3**
2.1 Campi di potenziale artificiale 3
3 Primo passo: descrizione operativa **4**
3.1 Campi di potenziale a griglia 4
3.2 Dati 4
3.3 Calcolo del potenziale 5
4 Secondo Passo: Concorrenza **5**
Versione semplificata
4.1 Implementazione dei satelliti 5
5 Terzo passo: ROS **6**
5.1 Installazione 6
Visualizzatore
5.2 Esempio di rover 7
Riferimenti bibliografici **7**

Introduzione

Immaginate di dover controllare una flotta di robot che si muovono sulla superficie del pianeta Marte (Fig. 1. L'obiettivo è raccogliere dei campioni individuati in una precedente missione. Ogni robot quindi riceverà la posizione di un campione e dovrà essere in grado di pianificare un percorso che lo porti a recuperare il campione cercando di evitare ostacoli sul percorso (ad esempio rocce o crateri) e cercando anche di evitare incidenti con altri rover. Il sistema dovrà essere intelligente, in grado di evitare di mandare contemporaneamente più robot sullo stesso campione e per questo i robot dovranno conoscere in ogni momento le posizioni degli altri rover.

Il vostro programma dovrà quindi risolvere alcuni problemi. Il primo è quello della pianificazione del moto del sistema robotico (Sec. 1). Vi verrà richiesto lo sviluppo di un algoritmo noto in letteratura come metodo dei campi di potenziale (Sec. 2) e vi verranno date le linee guida per lo svolgimento (Sec. 3). In una seconda fase dovrete essere in grado di estendere il problema creando diverse istanze dei robot che dovranno condividere le loro posizioni in modo da evitare collisioni. Inoltre i robot riceveranno gli obiettivi da un sistema esterno tramite un buffer di condivisione. Infine, come ultimo step, vi chiederemo di mandare le vostre traiettorie via ROS a dei robot simulati per mostrare l'efficacia degli algoritmi sviluppati al punto precedente. Le informazioni sugli ultimi due passi verranno fornite nelle prossime settimane a valle della trattazione dell'argomento a lezione.

1. Pianificazione del moto

La pianificazione del moto, anche conosciuta come pianificazione del percorso è un problema computazionale che ha come obiettivo individuare una sequenza di configurazioni valide per muovere un oggetto da una sorgente a una destinazione. Per esempio, considerate un robot mobile che deve muoversi all'interno di un edificio verso un obiettivo. Il robot dovrebbe

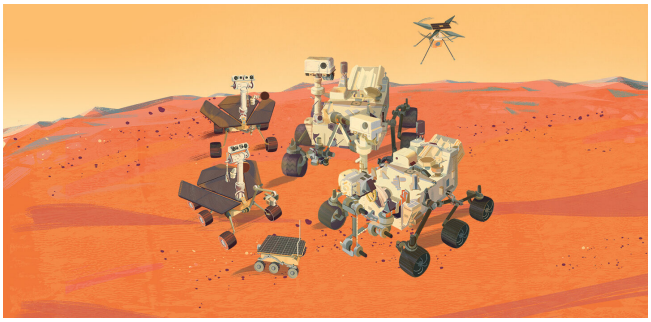

Figura 1. Evoluzione dei robot inviati su Marte.

eseguire il suo compito cercando, contemporaneamente, di evitare la collisione verso le pareti o il cadere dalle scale. Un algoritmo di pianificazione del moto prende la descrizione di questo compito come input, e produce i comandi che consentono al robot di raggiungere il suo obiettivo finale. Anche se il problema può assumere forme decisamente complesse (robot manipolatori a diversi gradi di libertà, veicoli non olonomi, ambienti parzialmente conosciuti) l'assegnamento si concentrerà su un problema semplificato a pochi gradi di libertà.

1.1 Concetti di Base

Prima di descrivere l'algoritmo che dovrete sviluppare, introduciamo la terminologia necessaria.

Un problema di base della pianificazione del moto ha come obiettivo calcolare il percorso continuo che unisce una configurazione iniziale (q_I) alla configurazione goal (q_{goal}), mentre evita la collisione con ostacoli conosciuti. Il robot e la geometria sono descritti in uno spazio bi- o tri-dimensionale noto come *workspace* (Fig. 2). Per rappresentare il moto del robot si preferisce, invece, vedere il percorso in uno *spazio delle configurazioni* di dimensione pari ai gradi di libertà del robot e quindi a dimensione solitamente più alta rispetto al workspace.

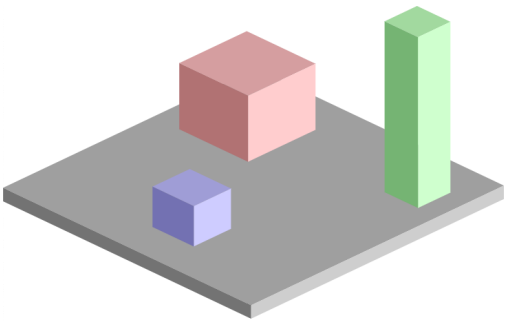

Figura 2. Esempio di workspace

Spazio delle configurazioni

Una configurazione descrive la posa del robot, e lo *spazio delle configurazioni* (*configuration space C*) è l'insieme di tutte le possibili configurazioni.

Iniziamo con un semplice esempio. Consideriamo di avere un robot di dimensioni infinitesimali (un punto) e che i suoi gradi di libertà siano limitati al traslare su un piano bidimensionale. In questo caso, il piano è il workspace del robot, per definire la sua posizione all'interno del workspace è sufficiente utilizzare due parametri, (x,y), e questo è una configurazione del robot. Anche lo spazio delle configurazioni sarà quindi bidimensionale, perché due sono i gradi di libertà del robot e quindi due i parametri da definire per individuarne la posizione.

A causa della presenza di ostacoli, non tutte le configurazioni possono essere raggiunte dal robot. L'insieme delle configurazioni che evitano collisioni con gli ostacoli vengono chiamate spazio libero (*free space*, C_{free}). Il complementare del C_{free} è lo spazio degli ostacoli (*obstacle space*, C_{obs}) o regione proibita. La Fig. 3 mostra lo spazio delle configurazioni per un robot puntiforme all'interno del workspace in Fig. 2.

Se il robot non è puntiforme, bisognerà considerare che alcune configurazioni non possono essere acquisite dal robot perché, in tal caso, colliderebbe con gli ostacoli. La Fig. 4 mostra un esempio, dove il C_{obs} include tutte le configurazioni che portano a una collisione con gli ostacoli.

Figura 3. Spazio delle configurazioni per un robot puntiforme. In bianco il C_{free} e in grigio il C_{obs}

Esempi di problemi più complessi

Il problema della pianificazione del moto può sembrarvi banale dagli esempi precedenti ma, in realtà, non è così perché calcolare lo spazio delle configurazioni diventa complicato all'aumentare dei gradi di libertà del robot e del numero degli ostacoli. Consideriamo, ad esempio, un robot non più puntiforme ma dotato di una forma e con un ulteriore grado di libertà, cioè la rotazione. In questo caso il workspace è ancora bi-dimensionale ma lo spazio delle configurazioni diventa invece $C = SE(2) = R^2 \times SO(2)$, dove SO(2) rappresenta lo spazio delle rotazioni 2D. Le dimensioni di C saranno quindi tre perché tre sono i parametri da specificare per identificare la posizione del robot, i.e. (x,y,θ), dove θ è la rotazione del robot.

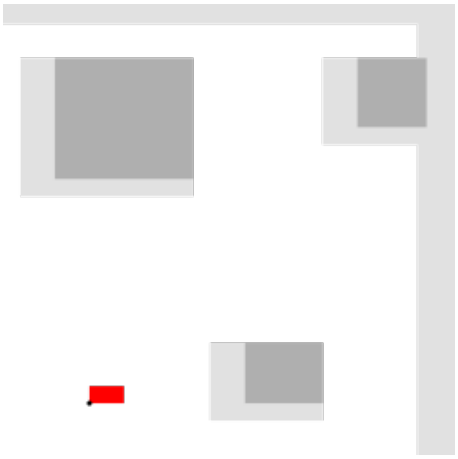

Figura 4. Spazio delle configurazioni di un robot rettangolare (in rosso) che può solo traslare. In bianco il C_{free} e in grigio il C_{obs}, il grigio scuro sono gli ostacoli, il grigio chiaro le configurazioni dove il robot toccherebbe gli ostacoli o uscirebbe dal workspace e pertanto non sono configurazioni accettabili

E le cose possono diventare ancora più complicate. Ad esempio, se considerate un robot planare a due gradi di libertà $\theta 1, \theta 2$ (Fig. 5) e supponete che ogni angolo possa muoversi da 0 a 360 gradi ottenete uno spazio delle configurazioni toroidale. Ogni configurazione del robot è un unico punto sul toroide e ogni punto sul toroide individua univocamente la posizione del robot.

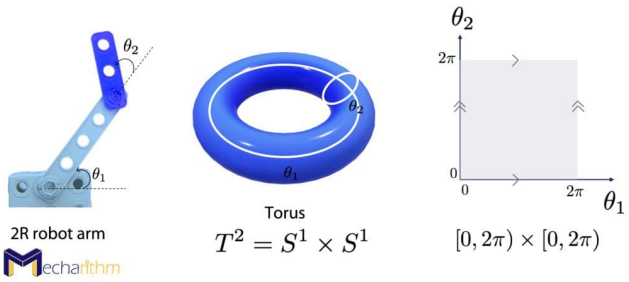

Figura 5. Spazio delle configurazioni di un robot planare a due gradi di libertà in due diverse rappresentazioni.

E le cose diventano ancora più complicate se mettiamo degli ostacoli all'interno di questo spazio. Fig. 6 mostra un esempio di uno spazio delle configurazioni bidimensionale (quindi ancora molto semplice) in presenza di alcuni ostacoli. Nello spazio delle configurazioni è indicato anche il percorso del robot (la sequenza di configurazioni) per passare dalla configurazione iniziale a quella finale senza collidere con gli ostacoli. Se volete provare a popolare un workspace con gli ostacoli e vedere come diventa lo spazio delle configurazioni potete provare a farlo qui: `https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml`.

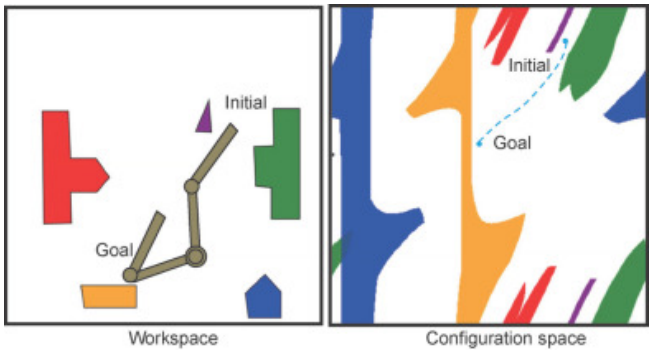

Figura 6. Spazio delle configurazioni di un robot planare a due gradi di libertà con presenza di ostacoli.

2. Metodo dei campi di potenziale

Nella prima parte dell'assegnamento risolverete il problema di base della pianificazione del moto. Dato un robot $\mathscr{A}$ e un modello degli ostacoli $\mathscr{O}$ e due configurazioni$q_I, q_{goal} \in C_{free}$ dovete calcolare una percorso $\tau : [0,1] \to C_{free}$ tale che $\tau(0) = q_I$ e $\tau(1) = q_G$.

Ci sono tre approcci principali per risolvere il problema di pianificazione del moto:

- combinatorio (pianificazione esatta)
- basati su campi di potenziale artificiale
- basato sul campionamento (pianificazione probabilistica o casuale)

Il primo approccio è adatto solo per problemi semplici che permettono di costruire C_{obs} e C_{free} in modo esatto, con l'aumentare dei gradi di libertà l'approccio diventa computazionalmente intrattabile. Gli algoritmi basati sui campi di potenziale sono efficienti, ma cadono preda del problema dei minimi locali. Gli algoritmi più usati sono basati sul campionamento perché sono molto spesso molto veloci nell'individuare un percorso.

Se foste interessati ad approfondire l'argomento, potete fare riferimento a [1] per una trattazione teorica.

2.1 Campi di potenziale artificiale

L'uso dei campi di potenziale artificiali è molto usato in ricerca per i suoi vantaggi quali la semplicità di implementazione e l'eleganza. Questo metodo permette inoltre di gestire ambienti dinamici, cosa non sempre facile per altri approcci, prendendo in considerazione lo stato attuale del workspace per influenzare il moto del robot. Il primo a suggerire l'idea di vedere il robot come un punto sotto l'influenza di un campo generato dalla configurazione obiettivo e dagli ostacoli nello spazio fu O. Khatib[2]. Due sono le forze coinvolte in questo metodo: una *forza repulsiva* generata dagli ostacoli e una *forza attrattiva* generata dall'obiettivo. Queste forze sono molto forti vicino all'ostacolo o al goal e hanno un effetto molto più debole, se non nullo, a grande distanza. La forza risultante (la somma di tutte le forze) è usato per determinare la direzione

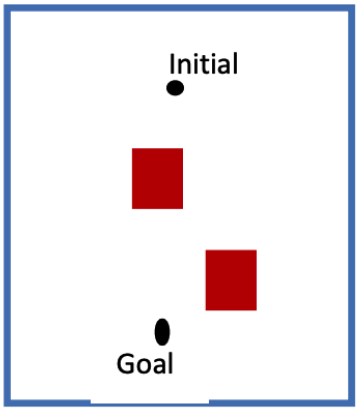

Figura 7. Spazio delle configurazioni con ostacoli (in rosso) e configurazioni iniziale e obiettivo.

che il moto del robot deve seguire per raggiungere l'obiettivo evitando le collisioni.

Campo attrattivo

Considerate la configurazione attuale di un robot q e la configurazione obiettivo q_{goal}. Il potenziale attrattivo può essere espresso in diverse forme:

conico: $U_{att}(q) = \zeta d(q, q_{goal})$

quadratico: $U_{att}(q) = \frac{1}{2}\zeta d^2(q, q_{goal})$

dove $d(q, q_{goal})$ rappresenta la distanza euclidea tra la configurazione attuale e la configurazione obiettivo e ζ è una costante che rappresenta un fattore di scala.

La forza attrattiva sarà quindi pari al gradiente negativo del potenziale. Nel caso di un potenziale quadratico diventa quindi pari a:

$$F_{att}(q) = -\nabla U_{att}(q) = \nabla(\frac{1}{2}\zeta d^2(q, q_{goal})) \quad (1)$$

$$= \frac{1}{2}\zeta \nabla d^2(q, q_{goal}) \quad (2)$$

$$= \zeta d(q, q_{goal}) \quad (3)$$

Campo repulsivo

Il campo di potenziale deve essere disegnato per spingere il robot lontano dagli ostacoli. L'idea però è che questo campo repulsivo sia forte intorno agli ostacoli, ma che il moto del robot non venga disturbato inutilmente quando l'ostacolo è lontano. Il potenziale repulsivo può pertanto essere definito come:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{D(q)} - \frac{1}{Q^*})^2, D(Q) \leq Q^* \\ 0, D(Q) \geq Q^* \end{cases} \quad (4)$$

dove η è una costante di scala che rappresenta il coefficiente di scala del campo repulsivo, $D(q)$ la distanza tra il robot e l'ostacolo e Q^* rappresenta la distanza massima di influenza. Da qui è possibile calcolare la $F_{rep}(q) = \nabla U_{rep}(q)$.

Funzione potenziale complessiva

Il potenziale complessivo può essere quindi rappresentato come:

$$U(q) = U_{att}(q) + U_{rep}(q) \quad (5)$$

e la forza complessiva diventerà quindi:

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) \quad (6)$$

$$F(q) = F_{att}(q) + F_{rep}(q) \quad (7)$$

La Fig. 7 rappresenta uno spazio delle configurazioni dove i punti sono le configurazioni iniziali e finali e i quadrati rossi rappresentano gli ostacoli. I campi potenziali corrispondenti sono rappresentati in Fig. 8

Il valore delle costanti può andare a modificare le curve dei potenziali (Fig. 9) e per questo potrebbe essere interessante esplorare come modificarli possa modificare anche il percorso del robot.

3. Primo passo: descrizione operativa

3.1 Campi di potenziale a griglia

In questo assegnamento svilupperete una versione discretizzata dell'algoritmo a campo di potenziale che rende decisamente più semplice il vostro lavoro.

Il vostro spazio delle configurazioni verrà pixellato in celle con una dimensione che definirete come parametro in ingresso. A ogni passo dell'algoritmo di pianificazione sarete quindi messi di fronte alla decisione di dove spostare il vostro robot e avrete 8 opzioni (Fig. 10).

Ogni cella ha un potenziale che la caratterizza in funzione degli ostacoli che la circondano e della distanza dalla cella obiettivo. Invece di calcolare quindi il potenziale dell'intero spazio delle configurazioni e la forza totale, potrete limitarvi a valutare il potenziale delle celle che circondano il vostro robot a ogni passo e scegliere di spostarvi nella cella a potenziale inferiore.

3.2 Dati

Il robot è un robot cilindrico di cui conoscete il raggio ed è olonomo. Questo vi permette di definire la configurazione del robot semplicemente con le coordinate del suo centro. Il robot si muove anche sempre in modo preciso così da evitare la correzione degli errori di posizione. I dati che definiscono il robot sono quindi:

- Raggio del robot [m] (se non riuscite a gestirlo, potete considerarlo puntiforme)
- Posizione iniziale del robot ([m], [m])

Il robot si trova all'interno di un ambiente senza barriere (così vi evitate i campi di potenziali dei bordi) che ha però un sistema di riferimento con una origine a cui tutte le posizioni fanno riferimento. L'ambiente viene quindi definito da:

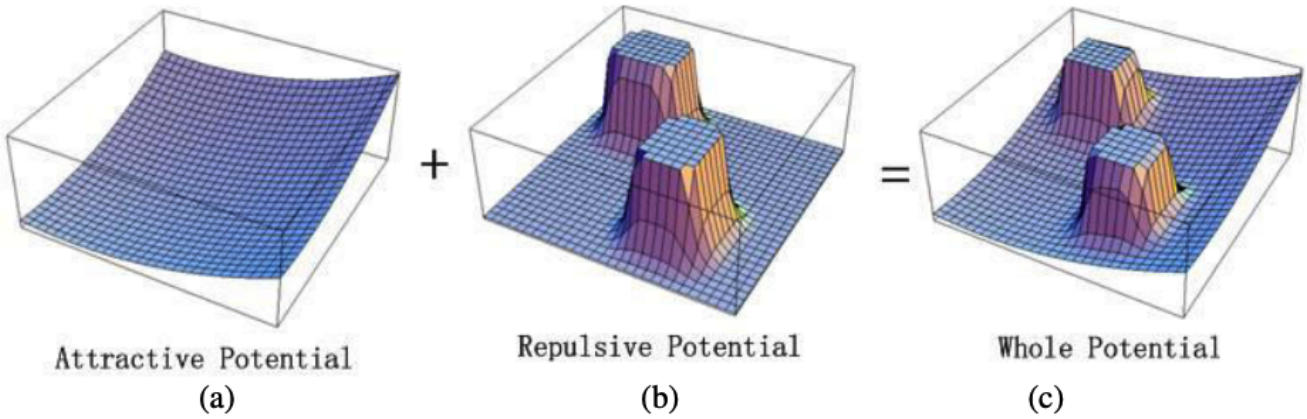

Figura 8. (a) il campo attrattivo senza ostacoli (b) il potenziale repulsivo con il valore più alto in corrispondenza dell'ostacolo (c) il potenziale complessivo come comnbinazione dei due campi precedenti

- la dimensione della cella della griglia per il calcolo del potenziale, la dimensione deve essere sempre un multiplo o un divisore del metro [m]
- gli ostacoli che sono presenti nell'ambiente. Per semplificare gli ostacoli sono sempre rettangolari, posti parallelamente agli assi del sistema di riferimento e di dimensioni multiple di metri. Se un ostacolo occupa anche parzialmente una cella, tutta la cella viene considerata parte del C_{obs}. Ogni ostacolo viene specificato indicando le coordinate di due angoli opposti, prima quello a coordinate $(min(x1,x2),min(y1,y2))$ e poi quelle a coordinate $(max(x1,x2),max(y1,y2))$.

Infine il problema richiede di definire:

- i valori delle costanti ζ e η
- la posizione goal da raggiungere ([m], [m])

3.3 Calcolo del potenziale

In ogni cella dello spazio delle configurazioni, il potenziale verrà calcolato come la somma del campo repulsivo e del campo attrattivo. Utilizzate per il calcolo del potenziale attrattivo la forma quadratica. Per calcolare il campo repulsivo e semplificare l'algoritmo considerate l'influenza del solo ostacolo più vicino.

Ricordatevi di non posizionare gli ostacoli troppo vicini perché il metodo di risoluzione basato sui campi di potenziale porta facilmente a minimi locali (Fig. 11) che rappresentano dei cul-de-sac che possono essere trattati solo con algoritmi più complessi.

4. Secondo Passo: Concorrenza

Due satelliti sono in grado di individuare la posizione dei campioni da recuperare che poi condividono con i robot. Lo scambio di informazioni avviene attraverso una coda di dimensione finita che contiene, al massimo $k*N$ campioni, dove N è il numero di robot disponibili e k un parametro a vostra scelta. É condizione necessaria che nessuna coordinata vada persa, inoltre ogni campione deve essere prelevato da un solo robot.

Ogni robot, una volta raggiunto l'obiettivo, va a leggere le coordinate di un altro campione da prelevare e va a seguire il percorso per raggiungerla utilizzando l'algoritmo sviluppato precedentemente. Per una soluzione più avanzata potete far scegliere al robot il campione più vicino tra quelli presenti in coda. Durante la pianificazione del percorso ogni robot conosce la posizione degli altri robot che devono essere considerati come ostacoli ed evitati.

Anche se non completamente realistico, si supponga che tutto il sistema si trovi su uno stesso server e quindi lo scambio di informazioni avvenga con memoria condivisa protetta da tecniche di sincronizzazione concorrenti.

4.0.1 Versione semplificata

Se vi trovaste in difficoltà nella soluzione del problema completo potete semplificarlo consegnando una soluzione con un solo robot.

4.1 Implementazione dei satelliti

Nel programma che simulerà il comportamento del satellite può tornarvi utile la possibilità di leggere da file le coordinate degli obiettivi. In questo modo se volete testare il vostro programma con sequenze diverse potete semplicemente cambiare il file di testo senza dover ricompilare il programma.

Nel listato 1 (e nel sito del corso) trovare un semplice codice che fa quello che vi serve.

```
#include <iostream>
#include <fstream>


void read_from_file(const std::string& filename) {
  std::ifstream infile{filename};

  while (!infile.eof()) {
    double x, y;
    infile >> x >> y;
    if (infile.eof() || infile.fail() || infile.bad()) {
```

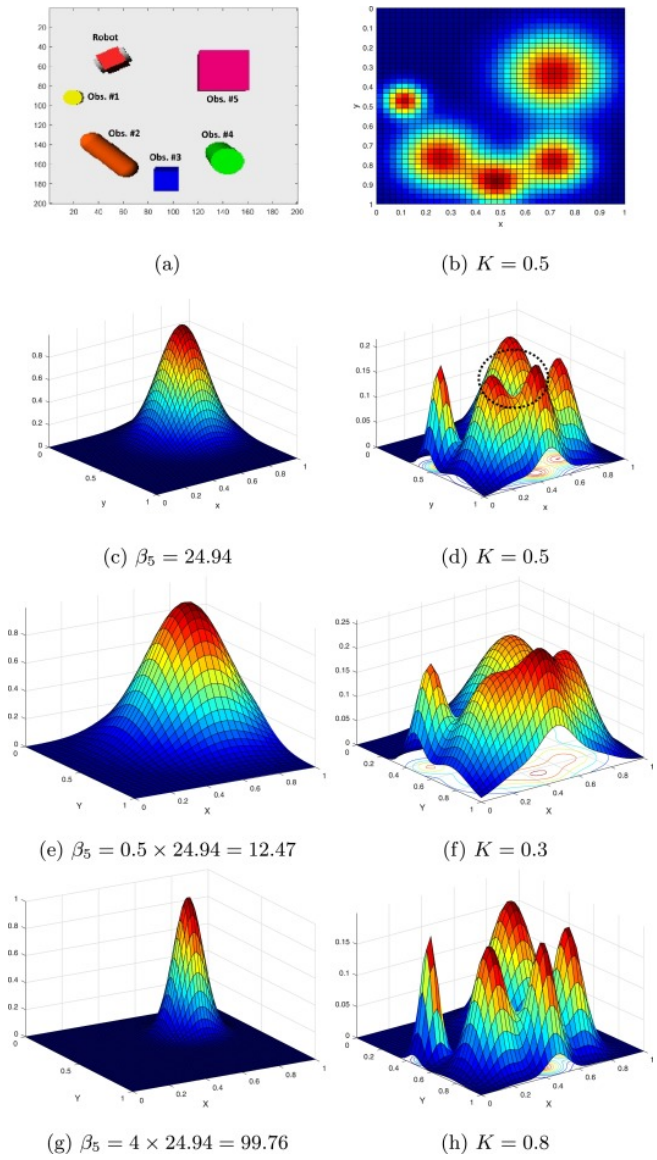

Figura 9. Influenza dei parametri η e ζ sul campo potenziale.

```
12        std::cerr << "Error in input or eof  \n";
13        break;
14      }
15      std::cout << x << " " << y << "\n";
16    }
17  }
18
19  int main() {
20    std::string filename{"sample_coordinates.txt"};
21    read_from_file(filename);
22
23    return 0;
24  }
```

Listing 1. Lettura da file

5. Terzo passo: ROS

Per l'ultimo passo di questo assegnamento abbiamo creato un programma in rviz che vi permette di visualizzare gli ostacoli

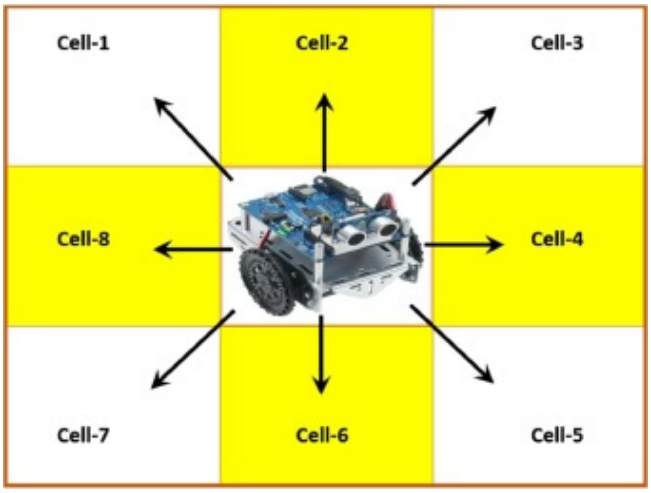

Figura 10. Possibili passi del robot.

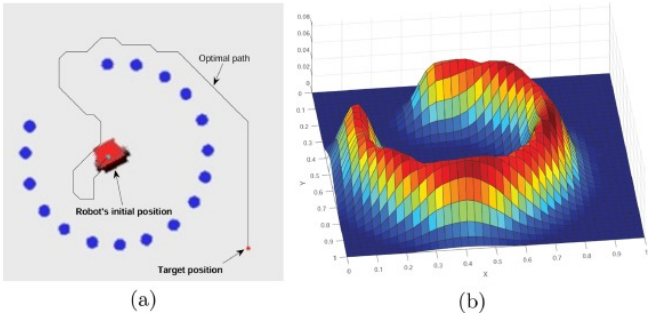

Figura 11. Minimo locale da qui è impossibile uscire.

e il percorso dei robot. Il vostro compito è integrare quello che avete svolto finora per pubblicare le posizioni

5.1 Installazione

Seguite la procedura per installare il software da cui partire:

1. Create una directory per il vostro workspace e, all'interno una directory `src` per contenere i vostri pacchetti.
2. Scaricate i pacchetti `rover` e `rover_visualizer` all'interno della directory `src`.
3. Se non lo avete già fatto settate l'ambiente ros:
   `source /opt/ros/humble/setup.bash`
4. Nella directory principale del workspace compilate i pacchetti con il comando:
   `colcon build --symlink-install`
5. Settate l'ambiente con:
   `source install/local_setup.sh`
6. Create un file `obstacle_positions.txt` con le posizioni degli ostacoli.
7. Dalla directory in cui si trova il file della posizione ostacoli lanciate il programma con:
   `ros2 launch rover_visualizer rover_launch.py`

Se vedete solo robot e griglia, ma non gli ostacoli, fate un zoom out per vedere la loro posizione.

5.1.1 Visualizzatore

Il pacchetto rover_visualizer contiene il codice del visualizzatore e *non deve essere modificato*. Se volete lanciare solo il visualizzatore con il subscriber delle posizioni del rover potete utilizzare il launch `rviz_launch.py`:

```
ros2 launch rover_visualizer rviz_launch.py
```

Il visualizzatore pubblica le posizioni dei rover che riceve sul topic `/rover`. Il messaggio da inviare è di tipo `rover_visualizer::msg::RoverPosition`:

```
int32 id
geometry_msgs/Point position
```

Listing 2. tipo del messaggio da inviare

In ogni messaggio va indicata la posizione aggiornata del rover (`position->x, position->y`) ed il numero univoco identificativo del rover da inserire nel campo `id` (per informazioni su `geometry_msgs/Pose2D` consultate: `https://index.ros.org/p/geometry_msgs/github-ros2-common_interfaces/`).

5.2 Esempio di rover

Se lanciate con il launch `rover_launch.py` oltre al visualizzatore viene fatto partire anche il nodo `rover` che è un semplice produttore di posizioni di due robot. Come vedrete dal codice abbiamo creato due rover piuttosto stupidi che, periodicamente, aggiornano la loro posizioni andato a compiere una linea retta. Il semplice esempio dei due rover vi servirà per prendere ispirazione su come preparare i messaggi e pubblicarli sul topic.

Partite da questo esempio ed aggiungete il vostro codice, aggiungendo il nome dei file che volete compilare all'interno del file `CMakeLists.txt` (per chi non ha seguito il corso di LII, scrivete sul forum se avete bisogno di un aiuto).

Infine, se volete modificare il nome del file in input che contiene la posizione degli ostacoli ed il raggio che definisce le dimensioni dei rover lo potete fare direttamente nei launch file (`xxx_launch.py`).

Ricordo infine che nei vari launch file e CMakeLists.txt abbiamo inserito un commento nelle righe che si possono/devono modificare, tutte le altre righe sarebbe meglio lasciarle come le trovate.

Riferimenti bibliografici

[1] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[2] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985.