Appunti di Robotica

Nikollaq Haxhi

12 gennaio 2016

Contents

1 Sistemi di controllo dei robot **5**
1.1 Navigazione . . . 5
1.2 Manipolazione . . . 5
1.3 Sistema di controllo . . . 5
1.3.1 Sfide nell'ingegneria dei sistemi di controllo per robot . . . 6
1.3.2 Problemi di progettazione . . . 6
1.3.3 Qualità dei robot - Simon's Ant . . . 7
1.3.4 Sistema di controllo di un robot autonomo . . . 7
1.4 Motor control . . . 7
1.5 Motion control . . . 8
1.5.1 Rover driver . . . 8
1.5.2 Target follower . . . 8
1.5.3 Cruise control . . . 8
1.5.4 Proximity control . . . 8
1.5.5 Robot arm motion control . . . 8
1.6 Motion planning . . . 8
1.7 Robot localization . . . 9
1.8 Map building . . . 9
1.9 Task planning . . . 9

2 Architetture dei componenti dei robot **11**
2.0.1 Significato dei blocchi . . . 11
2.1 Modulo software . . . 11
2.2 Componente software . . . 12
2.2.1 Connettori . . . 12

3 Ingegneria del software per la robotica **15**
3.1 Variabilità del sistema . . . 15
3.2 Domain specific Software Engineering . . . 15
3.2.1 Stili . . . 15
3.2.2 Tre fattori chiave . . . 16
3.3 Software product line (SPL) . . . 16
3.3.1 Challenges - Tecniche di creazione di una SPL . . . 18

4 Cinematica dei robot mobili su ruote **19**
4.1 Cinematica dei robot mobili . . . 19
4.1.1 Tipologie di ruote . . . 20
4.1.2 Tipologie di robot . . . 20
4.2 Centri di curvatura e rotazione . . . 21
4.2.1 Notazione . . . 21
4.2.2 Differential drive . . . 22
4.2.3 Triciclo . . . 22
4.2.4 Synchronous drive . . . 23
4.2.5 Omnidirezionale . . . 24
4.2.6 Automobile (Ackerman Steering) . . . 24
4.2.7 Carrello . . . 25
4.3 Odometria del differential drive . . . 25
4.3.1 Effector noise . . . 27

5 Controllo del moto **29**
5.1 Implementazione del controllo di moto . . . 29
5.1.1 Trajectory follower . . . 30
5.1.2 Pose controller . . . 30
5.1.3 Ramp generator . . . 30
5.2 Recovery trajectory . . . 30
5.3 Kinematic pose . . . 31

6 Obstacle avoidance **33**
6.1 Bug algorithm . . . 33
6.2 Vector field histogram . . . 33
6.3 Basic Curvature Velocity Methods . . . 34
6.4 Trajectory rollout algorithm (TRA) . . . 34
6.5 Dynamic window approach (DWA) . . . 34
6.5.1 Adattamento della traiettoria . . . 35
6.6 Oscillation Suppression . . . 36

7 Line following **37**
7.1 Rielaborazione dell'immagine . . . 37
7.1.1 Image convolution . . . 37
7.1.2 Riconoscimento dei bordi . . . 39
7.1.3 Trasformata di Hough sul piano . . . 39
7.1.4 Trasformata di Hough nello spazio . . . 40
7.1.5 Accumulatori . . . 40
7.2 Visual Servoing for Line Following . . . 41

8 Geometria 3D **43**
8.1 Geometria delle traslazioni . . . 43
8.2 Geometria delle rotazioni . . . 43
8.3 Angoli Roll - Pitch - Yaw (RPY) . . . 44
8.4 Angoli di Eulero I . . . 44
8.5 Angoli di Eulero II . . . 45
8.6 Coordinate omogenee . . . 45
8.7 Composizione di roto-traslazioni . . . 45

9 Mapping 2D **47**
9.1 Problemi . . . 47
9.1.1 Rumore di misurazione . . . 47
9.1.2 Problema della corrispondenza . . . 48
9.1.3 Ambiente dinamico . . . 48
9.2 Soluzioni - Probabilistic robotic mapping . . . 48
9.2.1 Approcci odierni . . . 48
9.3 Mappa geometrica 2D . . . 48
9.4 Mappa di occupazione 2D . . . 50
9.5 Mappa topologica 2D . . . 51

10 Robot Localization **53**
10.1 Probabilistic Map-Based Localization . . . 53
10.1.1 Passi della map-based localization . . . 53
10.1.2 Monte Carlo Localization using Particle Filtering . . . 54
10.2 Marker-based Localization and Navigation . . . 55
10.2.1 Triangolazione . . . 56
10.2.2 Triangolazione: posizione angolare del robot nota . . . 56
10.2.3 Triangolazione: posizione angolare del robot sconosciuta . . . 57

11 Mapping 3D **59**
11.1 Point Cloud . . . 59
11.2 Sistemi di misura . . . 59
11.2.1 Scanner 3D . . . 59
11.2.2 Time of flight scanner . . . 59
11.2.3 Scanner basati su triangolazione . . . 59
11.2.4 Structured light scanner (Microsoft Kinect) . . . 60
11.3 Visione stereoscopica . . . 60
11.3.1 Triangolazione . . . 60
11.4 Compressione dei dati . . . 61
11.4.1 Algoritmo Octree . . . 61
11.4.2 Segmentazione . . . 61

11.4.3 Generazione di meshes . . . 62

12 Simultaneous Localization and Mapping (SLAM) 63
12.1 Localizzazione . . . 63
12.2 Mapping . . . 64
12.3 SLAM . . . 64
12.4 Metodo Bayesiano approssimato per lo SLAM . . . 65
12.4.1 Extended Kalman Filter (EKF) . . . 65
12.5 SLAM: Stato aumentato . . . 66
12.6 Conclusioni EKF & SLAM . . . 67

13 Monocular SLAM 69
13.1 Algoritmo ICP (Iterative Closest Point) . . . 69
13.2 Optical Flow . . . 69
13.2.1 Stima coarse2fine . . . 70
13.3 Conclusioni . . . 71

14 Automated Planning 73
14.1 State Transition System . . . 73
14.2 Pianificatore vs Scheduler . . . 74
14.3 Domain specific . . . 75
14.4 Domain indipendent . . . 75
14.4.1 Classical Planning . . . 75
14.5 Pianificazione HTN (Hierarchical Task Networks) . . . 77
14.5.1 Simple Task Network (STN) Planning . . . 77
14.6 Markov Decision Processes . . . 77
14.7 Partially observable Markov Decision Process (POMDP) . . . 78

15 Meccanica dei manipolatori 79
15.1 Giunti . . . 79
15.2 Link . . . 79
15.3 Gradi di libertà . . . 80
15.4 Spazio di lavoro . . . 80
15.5 Esempi di bracci . . . 80
15.5.1 Robot cartesiano (TTT) . . . 80
15.5.2 Robot cilindrico (RTT) . . . 80
15.5.3 Robot sferico (RRT) . . . 81
15.5.4 Robot articolato (RRR) . . . 81
15.6 Polso di Robot . . . 81

16 Cinematica dei manipolatori 83
16.1 Cinematica diretta . . . 83
16.1.1 Denavit Hartemberg . . . 83
16.2 Cinematica inversa . . . 85
16.2.1 Molteplici soluzioni . . . 86
16.2.2 Metodi di soluzione . . . 86
16.2.3 Esempio . . . 86
16.2.4 Metodo di Paul . . . 86
16.2.5 Sostituzione di variabile . . . 87
16.2.6 Caso particolare con 3 giunti rotatori e un polso a 3 gradi di libertà . . . 87
16.3 Cinematica differenziale . . . 87
16.3.1 Esempio . . . 87
16.3.2 Jacobiana per piccole variazioni . . . 88
16.3.3 Piccole rototraslazioni . . . 88
16.3.4 Relazione tra $\mathbf{\Delta}$ e $\mathbf{J}$ . . . 89
16.3.5 Cinematica differenziale inversa - Jacobiana inversa . . . 89
16.4 Metodi iterativi per la Cinematica Inversa . . . 90
16.4.1 Metodo del gradiente . . . 90
16.4.2 Jacobian Inverse Kinematics . . . 90

16.4.3 Inverting the Jacobian Matrix . . . 91
16.4.4 Problemi della Inverse Kinematics . . . 91

17 Motion planning 2D **93**
17.1 Completezza del pianificatore . . . 93
17.2 Spazio delle configurazioni . . . 93
17.2.1 Braccio robot - spazio delle configurazioni . . . 94
17.2.2 Il robot non è un punto . . . 94
17.3 Minkowski's sum . . . 95
17.4 Estensione del problema . . . 95
17.5 Approcci alla pianificazione dei percorsi . . . 95
17.5.1 Global planning . . . 95
17.5.2 Local planning . . . 95
17.5.3 Algoritmi pratici . . . 95
17.6 Decomposizione in celle . . . 96
17.6.1 Decomposizione esatta . . . 96
17.6.2 Decomposizione approssimata . . . 96
17.7 Campi potenziali . . . 97

18 Probabilistic roadmaps **99**
18.1 Probabilistic Road Map (PRM) . . . 99
18.1.1 Learning phase . . . 100
18.2 Query phase . . . 101
18.3 Kinodynamic Planning . . . 101
18.4 Sampling strategies . . . 102
18.4.1 The Gaussian Sampling Strategy . . . 102
18.4.2 The Bridge Test for Sampling Narrow Passages . . . 102
18.4.3 Bridge Test Hybrid . . . 103

19 Controllo assi **105**
19.1 Spazio cartesiano . . . 105
19.2 Spazio giunti . . . 105
19.2.1 Calcolo della traiettoria singolo asse . . . 106
19.2.2 Calcolo della traiettoria con una velocità costante in mezzo . . . 106
19.2.3 Controllo cinematico di un asse . . . 107

20 Controllo in forza **109**

21 Collision detection **111**
21.1 Algoritmo . . . 111
21.1.1 Creazione dell'albero di palle . . . 111
21.1.2 Branch and bound . . . 111
21.1.3 Collision Detection . . . 111

1 Sistemi di controllo dei robot

Un robot deve essere in grado di *navigare* nell'ambiente che lo circonda e *manipolare* oggetti con l'utensile sul suo braccio (gli utensili montati sui suoi bracci).

1.1 Navigazione

Navigazione si intende il processo di pianificazione, monitoraggio, coordinamento e controllo del movimento del veicolo per andare da un punto di origine A a un punto di destinazione B.

Affinchè il robot possa navigare nell'ambiente è necessario che si localizzi e quindi capisca dove si trova in un determinato momento, grazie all'analisi dell'ambiente (ostacoli) che lo circondano (usando videocamere, sensori laser, sonar a ultrasuoni, tag marker, ecc).

La navigazione, noto il posto in cui si trova il robot, usa un pianificatore di moto che dice al robot come muoversi da A a B evitando gli ostacoli (PIANIFICAZIONE DEL MOTO). Quest'ultima inoltre è utilizzate per compiere varie operazioni, tra cui: trasporto, recupero oggetti, spostare oggetti, pulizia, tagliare l'erba, ecc.

1.2 Manipolazione

Oltre alla navigazione il robot (quando dotato), può anche manipolare oggetti attraverso un utensile sul suo braccio (gli utensili montati sui suoi bracci).

Manipolazione si intende il processo di pianificazione, monitoraggio, coordinamento e controllo del movimento del braccio robot articolato con montato un utensile alla sua estremità.

Destrezza la capacità dei motori delle mani/dita di prendere, raccogliere e manipolare un oggetto in modo complesso.

La manipolazione condivide delle informazioni anche con la navigazione. Infatti se il braccio è vincolato a una base mobile deve sapere (quando la base si muove), che ostacoli ci sono attorno basandosi sulla posizione nella mappa in cui si trova.

Nel caso il movimento del braccio del robot è *ripetuto* più volte, si può renderlo ottimo in modo da non avere sprechi in termini di tempo e di risorse energetiche.

1.3 Sistema di controllo

Il sistema di controllo del robot consiste in un unione e interazione delle capacità di navigazione e manipolazione secondo un'architettura di controllo specifica.

L'utente interagisce con i robot usando un linguaggio di alto livello per specificare i task.
Il robot decompone la missione in tanti task elementari tali per cui sia più facile da ottenere il risultato finale. Infatti l'esecuzione dei task consite in azioni di manipolazione/navigazione semplici che possono essere eseguite concorrentemente o sequenzialmente.

Per esempio la consegna medicine

1. Recuperare la mappa dell'ospedale da un server remoto;
2. Identificare la poszione attuale del robot nella mappa;
3. Pianificare il percorso dalla posizione attuale alla stanza X del paziente $ABCD$;
4. Muoversi (navigare) lungo il percorso pianificato precedentemente;
5. Prendere le medicine dal carrello a bordo e appoggiarle sul tavolo vicino al paziente.

Continuando con l'esempio di sopra:

- Potrebbe capitare che il robot abbia bisogno di iterare diverse volte tra il task 3 e il task 4 prima di raggiungere il paziente, perchè per esempio il percorso pianificato non è percorribile.
- Il task 4 potrebbe richiedere di aprire la porta della stanza del paziente, ciò richiede abilità di manipolazione.

1.3.1 Sfide nell'ingegneria dei sistemi di controllo per robot

Ci sono 3 problematiche principali:

- Risorse limitate (potenza di calcolo, memoria, autonomia batterie), quindi è necessario ottimizzarle affinchè il robot possa eseguire la sua misisone/task in modo autonomo.
  - Per ridurre l'onere computazionale del robot, si interconnettono in rete diversi calcolatori così da distribuire il carico. Si parla infatti di nodi del robot in rete (locale) che interagiscono tra loro.
- L'ambiente può contenere vincoli. Quindi i task del robot hanno dei vincoli, che siano temporali o spaziali allora è necessario pianificare l'attivita da svolgere in modo da rispettare i vincoli. Se qualcosa non funziona bisogna ri-pianificarlo immediatamente.
- I sensori a bordo del robot (o anche attorno a esso), sono soggetti a errori e incertezza. Quindi il robot deve essere in grado di rispondere/adattarsi immediatamente (entro certi limiti di tempo) a possibili failure/errori delle misurazioni effettuate. Il robot deve essere in grado ri rispondere *real-time* agli stimoli che gli vengono dall'ambiente esterno.
  - *e.g. rinconscimento di un eventuale ostacolo lungo il percorso pianificato (libero da ostacoli).*

1.3.2 Problemi di progettazione

Quando si realizza un robot bisogna sfruttare al meglio le risorse che ha a bordo, basandosi sull'obiettivo finale per cui il robot sarà utilizzato.

Quindi bisogna bilanciare le prestazioni del robot per avere una determinara autonomia.
Nella scelta dei sensori e degli attuatori bisogna considerare il fatto che sono limitati nelle prestazioni anche nelle capacità. *e.g. telecamera: risoluzione limitata, non funziona se non c'è luce.*

Sistemi ridondanti sono sistemi progettati in modo da essere robusti nelle varie condizioni non nominali.

- *localizzazione*
  - sensori IR
  - sensori ultrasuono
  - telecamera + telecamera IR
  - sensore scanner laser

Ma quale dei tanti sensori ha ragione, se usati tutti contemporaneamente?

La risposta si basa sui modelli probabilistici dei sensori.

- $p(telecamera) = 0.9$ mentre la $p(laser) = 0.3$.

Lo stesso procedimento viene seguito anche sugli attuatori.

- non è detto che se una ruota gira per $1\,m$ allora il robot si è mosso di $1\,m$, se il robot si muove su una superficie qualsiasi (dosso).

Si pesa pertanto l'azione degli attuatori con modelli probabilistici, e si usano tecniche simili perchè non si può costruire un modello esatto dell'ambiente circostante, tale da avere il controllo ottimo.
Il robot deve svolgere due cose nel suo sistema di controllo (SdC)

- capire/gestire la sua struttura meccanica;
- riconoscere/creare l'ambiente che lo circonda.

I SdC sono tutti molto simili (uguali) fra loro, l'unica cosa che li differenzia è l'ambiente in cui verranno usati. Quindi si ha della variabilità: struttura, ambiente, compito.

Se si dovesse creare da zero un SdC costerebbe tantissimo.
Bisogna progettare i SdC tale da differenziare quali sono i task generici e quali quelli particolari. In questo modo si possono riusare alcune parti generiche in altri progetti (VARIABILITA').

1.3.3 Qualità dei robot - Simon's Ant

Intelligenza L'abilità di muoversi nell'ambiente e mantenere delle interazioni dinamiche utili/adeguate con esso, senza l'intervento dell'uomo.

Per fare ciò il robot ha bisogno di avere delle capacità cognitive (percezione, pianificazione, monitoraggio, coordinamento e controllo), così da svolgere i task utili/complessi in modo sicuto e affidabile.

Si parla così di robot autonomi.

Embodiment L'abilità di comprendere di possedere un corpo avente un certo ingombro e soggetto a particolari limiti, attraverso il quale il robot può interagire con il mondo attorno a lui.

Le interazioni avvengono attraverso attuatori e sensori.

Situatedness La proprietà di esistere in un ambiente complesso, non strutturato e dinamico che influisce fortemente sul comportamento del robot.

Infatti i robot non sono come i videogiochi, in tutto è noto e ben definito a priori, ma non conoscono l'ambiente in cui si trovano:

- ***non strutturati***: l'ambiente non è stato progettato per rendere il lavoro del robot più semplice;
- ***dinamici***: le condizioni ambientali possono cambiare inaspettatamente (illuminazione, ostacoli imprevisti, oggetti in movimento, ecc.);
- ***parzialmente osservabili***: lo spazio di movimento è molto più ampio dello spazio di osservazione;
- ***incerti***:
  - i sensori sono imperfetti, le informazioni dipendono dalle condizioni ambientali
  - l'effetto delle azioni degli attuatori non è deterministico;
- ***inaccessibili all'uomo***: ambienti pericolosi, irraggiungibili.

1.3.4 Sistema di controllo di un robot autonomo

Nell'elaboratore sono presenti vari strati software che scompongono il task in azioni attuabili dai motori.

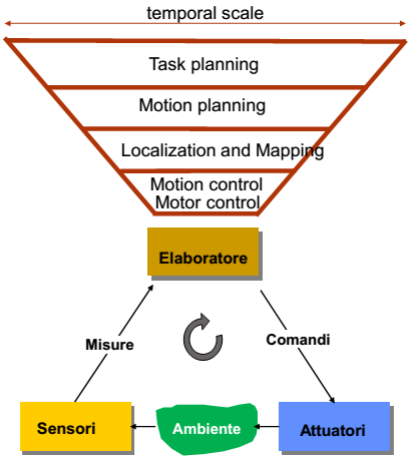


Figura 1: Sistema di controllo di un robot autonomo

1.4 Motor control

Il robot ha uno strato di controllo a basso livello in cui ci si occupa del controllo dei motori. Questo task è quello che ha una frequenza di esecuzione maggiore rispetto ad altri. Si hanno diverse tipologie di controllo: in velocità, in posizione, in impedenza.

Il ciclo viene eseguito in modo periodico (operazione real-time che di deve sempre completare entro il periodo), mentre il segnale di cambio velocità è asincrono e avviene in momenti non stabiliti a priori.

1.5 Motion control

Ci si occupa del movimento della struttura del robot. Quindi si sale di un livello e si astrattizza non considerando più direttamente l'hardware a bordo del robot (motori), ma l'intera struttura.

1.5.1 Rover driver

Si danno dei comandi di Twist $(\mathbf{v}, \omega)$ al robot per muoversi e, grazie agli encoder montati sui mototi delle ruote, fornisce informazioni sull'odometria $[O_x, O_y, \theta, V_x, V_y, \omega]$. Ovvero una stima della posizione del robot basato sulle velocità applicate alle ruote.

1.5.2 Target follower

Si specifica al robot, non più il comando di twist, ma la posizione finale che deve raggiungere. In questa situazione il robot deve navigare verso il target passando ovviamente da dei *waypoints*, perchè ci sono infiniti modi per raggiungere una posizione sul piano.

Percorso successione di punti nello spazio. Sequenza di configurazioni dei giunti di un robot in un particolare ordine.

Traiettoria percorso in cui si specifica anche la tempistica con cui bisogna completare ogni segmento. La traiettoria quindi aggiunge un informazione in più la percorso, il tempo. Dice non solo dove si deve trovare il robot ma anche in quale istante temporale.

$$TRAIETTORIA \neq PERCORSO$$

Per conoscere la traiettoria è necessario conoscere l'ambiente, quindi avere una mappa a disposizione.

1.5.3 Cruise control

Misurando la distanza della macchina che precede si è in grado di regolare di conseguenza la velocità di crociera.

Utilizzando quest'informazione è possibile anche effettuare dell'obstacle avoidance.

1.5.4 Proximity control

Il robot si muove per esempio lungo un muro, quindi è in grado di costeggiare l'ostacolo correggendo la propria traiettoria grazie alle informazioni che forniscono i sensori di prossimità a bordo.

1.5.5 Robot arm motion control

Si controlla il movimento di un braccio robotico nello spazio affichè possa raggiungere gli obiettivi prefissati.

1.6 Motion planning

Si cerca un percorso libero ostacoli che deve percorrere un corpo rigido o articolato (robot) nell'ambiente che lo circonda. Il robot conosce la mappa dell'ambiente in cui si deve muovere e anche il suo ingomboro e cinematica.

Alla fine il piano restituisce, se esiste, un insieme di waypoint attraverso cui il robot deve passare per raggiungere l'obiettivo.

Trajectory generator genera in output la traiettoria per raggiungere la destinazione, basandosi su una mappa dell'ambiente in cui si muove il robot. L'output viene dato al *trajectory adapter*.

Trajectory adapter genera nuove traiettorie in caso di eventi imprevisti (*e.g.* ostacoli) nella traiettoria, precedentemente definita. Prende in input la mappa e l'odometria e manda l'output al *target follower*.

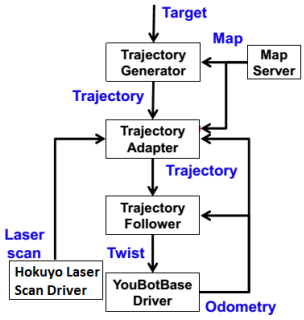


Figura 2: Sistema di controllo per l'inseguimento della traiettoria di una base mobile.

1.7 Robot localization

Il robot può localizzarsi in due modi: localizzazione assoluta e relativa. Nella localizzazione assoluta il robot capisce dove si trova nell'intero ambiente in cui è. In quella relativa invece stima una serie di spostamenti e si localizza nella mappa.

1.8 Map building

Il robot attraverso i sensori che ha montato a bordo costruisce una mappa dell'ambiente che lo circonda. Tale mappa può essere di due tipi: mappa di occupazione oppure mappa topologica.

Il robot potrebbe ricevere in input anche una mappa geometrica (fatta dall'architetto) e usarala per le vaire operazioni che deve svolgere.

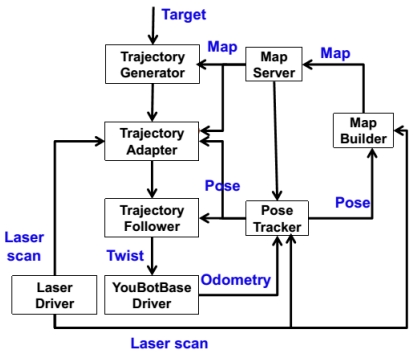


Figura 3: Sistema di controllo per il mapping dell'ambiente con una base mobile.

1.9 Task planning

Si pianifica l'insieme delle azioni che il robot deve seguire in sequenza/parallelo (se sowftware) affinchè possa raggiungere il suo obiettivo.

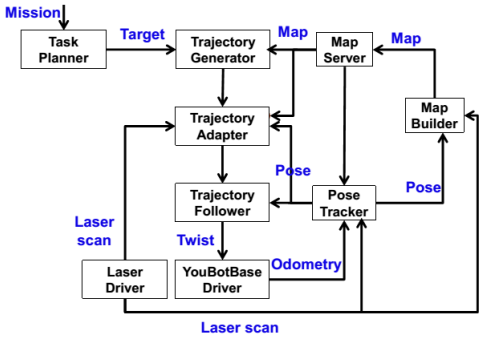


Figura 4: Sistema di controllo per il mapping dell'ambiente con una base mobile.

2 Architetture dei componenti dei robot

Sia definito lo schema di controllo di un robot mobile, dove ogni blocco è un modulo software che esegue operazioni e calcoli e offre un'interfaccia per comunicare con gli altri blocchi tramite pubblicazione di messaggi e sottoscrizioni a messaggi di altri blocchi.

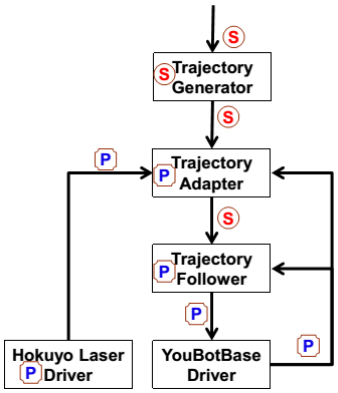


Figura 5: Sistema di navigazione di un robot mobile.

Si definiscono i seguenti eventi

- **periodico**: il task (l'informazione) è eseguito (ricevuta) periodicamente dal sistema.
- **sporadico**: non è possibile prevedere quando arriverà un'informazione esterna al blocco, dipende dall'ambiente.

Esempi

- il robot, per questioni di robustezza, riceve continuamente il messaggio di Twist anche quando è già fermo, per stare fermo. Questo perchè in caso di guasti il robot non si deve muovere e recare danni all'ambiente circostante.
- se il driver dei motori non ricevono i messaggi di Twist entro certi timeout allora il robot interrompe il movimento e si ferma.

2.0.1 Significato dei blocchi

I blocchi possono significare due cose:

- moduli software:
  - mottono in evidenza le caratteristiche delle strutture dati del modulo (usano gli *UML class diagram*).
- componenti software
  - hanno il significato di entità software in esecuzione, si evidenziano proprietà temporali del blocco (periodico/sporadico) (usano gli *UML component diagram*).

2.1 Modulo software

Nel caso dello youbot uso la classe che gestisce le ruote svedesi, ma potrei avere anche delle altre tipologie di ruote. Quindi si ha l'interfaccia ruota, e si sviluppano i vari controlli dei motori in base alle ruote che sono state montate sul robot.

Ogni blocco ha un suo *main()* all'interno, quindi è un processo in esecuzione sul calcolatore (si usano i thread POSIX per gestire i processi).

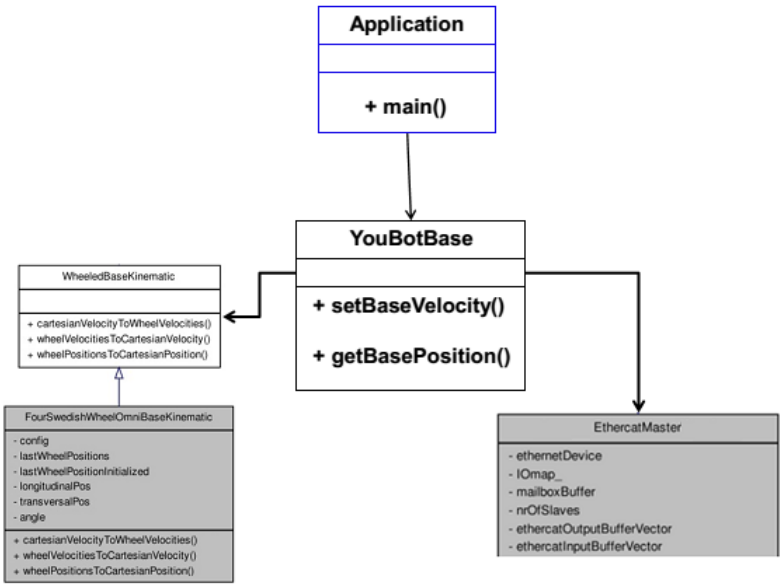


Figura 6: Modulo software.

2.2 Componente software

All'interno del sistema di controllo di un robot esistono diversi moduli che dialogano tra di loro attraverso aree di memoria condivise, accessibili in mutua esclusione.

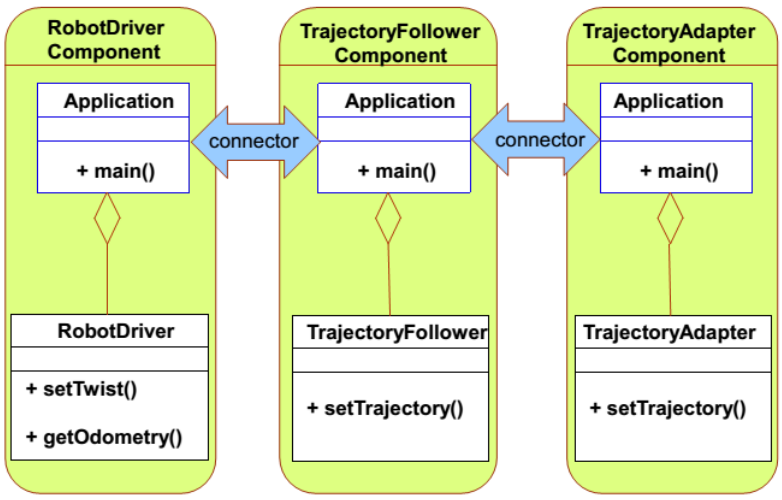


Figura 7: Componente software.

Le «capsule» in figura 7 rappresentano i componenti. Ogni componente può risiedere su una macchina (computer) diverso.

Il componente quindi è un applicazione in esecuzione su un calcolatore, in grado di comunicare con il mondo esterno attraverso lo scambio di messaggi.

2.2.1 Connettori

Il connettore è un componente elementare che permette la comunicazione tra componenti software.

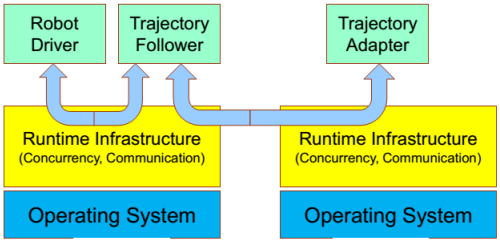


Figura 8: Connettori.

La struttura *runtime* simula la memoria condivisa. Quindi è come se fosse una memoria distribuita su più macchine.
Ogni connettore ha un suo protocollo

- client-server: il client invoca un servizio al server
  - one way: invocazione asincrona (no attesa di risposta).
  - deferred response: client attende la risposta dal server
  - request with callback: server conosce client e quando un'operazione è completa allora invoca una funzione di callback del client che gestisce la rispota del server.
- publish-subscriber: chi si connette, si registra a un *topic* e viene avvisato ogni volta che si pubblicano dati su quel *topic*.

Il metodo più usato è quello basato sul publish-subscriber (ROS, DDS, ROS 2, ecc). In questo modo non si devono avvisare uno ad uno i componenti che hanno chiesto il servizio, ma tutti in un colpo solo.

ROS (Robotic Operating System) è un runtime infrustructure che permette una modularità molto forte del sistema.

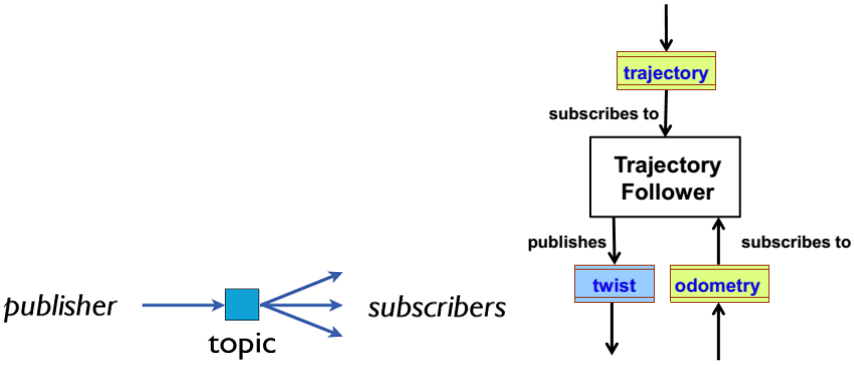


Figura 9: Nodo ROS.

3 Ingegneria del software per la robotica

3.1 Variabilità del sistema

La variablità è un fenomeno molto frequente ed è difficile da gestire per via di:

- cambio dell'ambiente di lavoro del robot
- aggiornamento della tecnologia, richiede l'aggiornamento del sistema affinchè funzioni come prima o addirittura meglio.

E' necessario pensare che il sistema software sia flessibile alle modifiche che si vogliono apportare.

Bisogna allora progettare il software tale da poterlo utilizzare per svolgere diversi task (che hanno qualcosa che gli accomuna) in contesti diversi (e.g. trajectory planning: per spazi aperti, per spazi stretti, per spazzi con ostacoli).

$$\Longrightarrow \textit{SISTEMA CONFIGURABILE PER UNA FAMIGLIA DI AZIONI.}$$

Ogni blocco del sistema, contiene degli algoritmi di controllo di diverse parti/componenti hardware e software del robot.

Allora se si hanno tanti blocchi con diversi algoritmi che svolgono la stessa funzione, posso provarli tutti sul sistema (per vedere il loro esito), a patto che il sistema sia modulare e che i blocchi vengano prima configurati, secondo determinate specifiche, e poi usati.

Il sistema di controllo dipende dalle funzionalità del robot, dalla sua struttura, da dove verrà utilizzato e dalla piattaforma software utilizzata (ROS, DDS, ROS 2, RT-Middleware).

3.2 Domain specific Software Engineering

Nella classica ingegneria del software si parte sempre dall'analisi dei requisiti per costruire il sistema *de novo*, il che comporta degli sforzi enormi e perdita di tempo.

Pertanto si può raggiungere lo stesso obiettivo percorrendo diverse strade.
L'idea è quella di acquisire informazione su soluzioni a problemi comuni.

Allora in teoria si portebbero implementare solo le differenze, tra quello che è il nuovo sistema e quello dei vecchi sistemi, dei vari problemi.

Nel domain specific si parte sempre dall'analisi del dominio in cui si usarà il software e si ottiene come risultato dell'analisi: chi lo usa, come lo usa e quali sono i vincoli.

3.2.1 Stili

- **architecture-based**
  - dato un problema si sceglie da un insieme di possibili stili architetturali adatti al nostro problema. Si parte da qui per lo sviluppo della soluzione al problema particolare.
- **domain-based**
  - si ritagliano i problemi e le soluzioni in sottoinsiemi (famiglie), quindi si parte da una famiglia di problemi e si arriva a una famiglia di soluzioni.
    - ▷ + è piccolo il dominio, + è specifica la soluzione
    - ▷ + è ampio il dominio, - è specifica la soluzione (è generale)

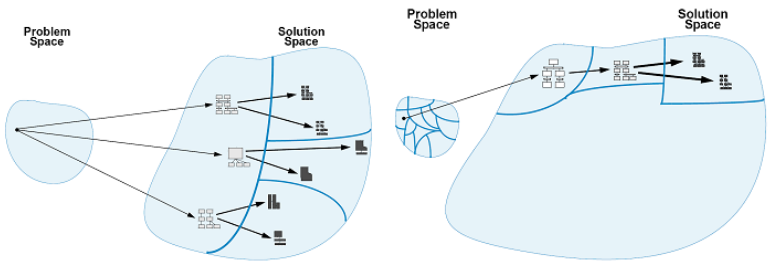


Figura 10: Insiemi di problemi e soluzioni architechture-based (a sinistra) e domain-specific (a destra).

3.2.2 Tre fattori chiave

Si mappano le regioni dello spazio del problema in DSSA (Domain Specific Software Architectures), queste vengono definite in architetture specifiche che vengono poi implementate.

- Domain
  - definisce il tipo di problema che risolve e viene descritto dal domain model, cioè un insisme di artefatti che descrivono le informazioni sul dominio (requisiti, funzionalità...), e permette di standardizzare la semantica e terminologia riguardo il problema.
- Tecnologia
  - definisce la tecnologia usata per risolvere il problema(tecniche e algoritmi a disposizione).
- Business
  - obbiettivi economici che hanno portato a questa soluzione.

L'intersezione dei tre fattori fornisce una product line architecture per un determinato DSSA.

Come nel mondo automobilistico, si definisce una linea di produzione di robot configurabili a seconda del dominio di lavoro.

Domain model comprende i requisiti e le funzionalità che la soluzione deve avere, serve per standardizzare la descrizione di problemi dello stesso tipo.

3.3 Software product line (SPL)

E' l'unione di:

- architettura di riferimento usata per un determinato dominio
  - una architettura di riferimento è una architettura in cui sono evidenziati le cose che possono essere modificate per adattarla all'applicazione.
- delle librerie riusabili dal dominio applicativo.
- dei metodi di configurazione per adattare la soluzione a una determinata applicazione.

Si può comporre in modo modulare aggiungendo o eliminando componenti a seconda delle necessità. Quindi una SPL è un insieme di sistemi che hanno molto in comune e quindi usare il metodo di SPL per creare un software è più economico perchè consiste principalmente nell'integrazione e nella personalizzazione di software già fatto.

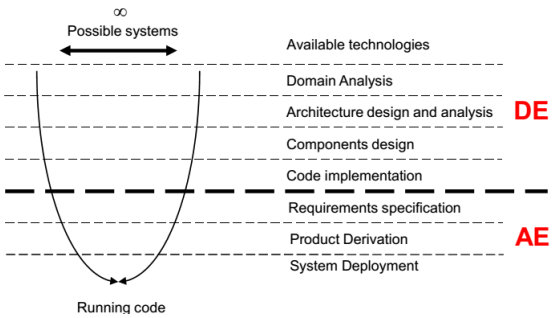


Figura 11: Software Product Line approach.

Come si nota in figura 11 il prolema è diviso in due parti:

- **Domani Engineering** (DE): rappresenta una famiglia di soluzioni per una certa famiglia di problemi.
- **Application Engineering** (AE): rappresenta il problema particolare che si vuole risolvere.

Component Repository Posto dove sono salvati i vari componenti della SPL fatti e che possono essere riusati. Qui si possono salvare anche diverse versioni dello stesso componente o anche diversi modi per implementarlo.

I vari componenti possono essere contrassegnati da dei vincoli:

- **Requires**, quando un componente per essere usato ne deve usare un altro.
- **Mandatory**, quando deve essere usato per forza.
- **Optional**, quando può anche non essere usato
- **Alternative**, quando se si usa un componente si può evitare di usarne un altro e viceversa

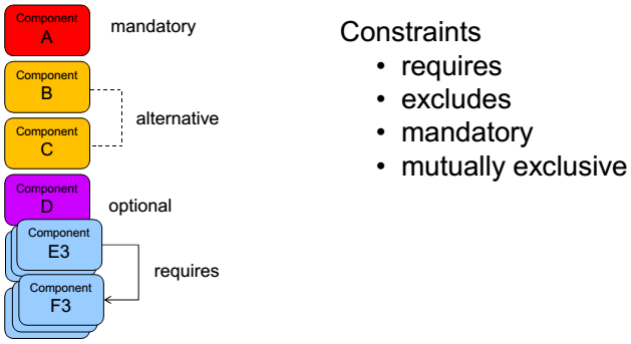


Figura 12: Vincoli del repository a componenti.

Software Product Line Architecture (SPLA) Descrizione delle architetture che collega i vari componenti del repository mettendo in evidenza i punti di variazione, descritti anche nel repository.

SPL Life Cycle per costruire una SPL è necessario:

- Un esperto di robotica individua i punti in comune e di variabilità per un determinato dominio, e produce un albero detto *domain feature model* che specifica i punti di variazione dei componenti.
  - Esempio:
    - ▷ Robust navigation
      - → Robot che può essere uno youbot o un pioneer.
      - → Camera position che può essere sul fronte o sul retro.
      - → ∗ ...

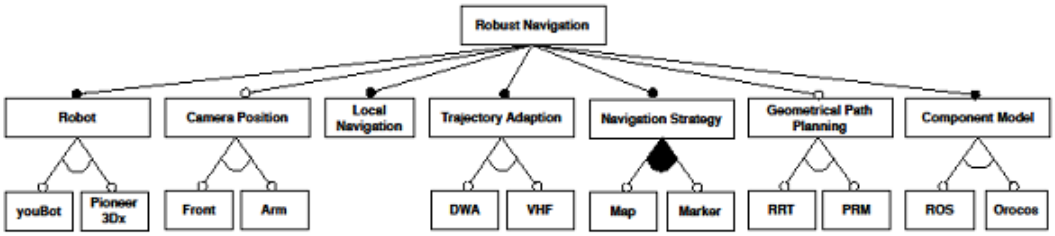


Figura 13: Domain feature model.

- Un ingegnere del software definisce una architettura usabile e abbastanza flessibile per implementare il software, e implementa i vari componenti mettendoli nel repository con i vari vincoli.
- Qualcuno integra tra di loro i vari componenti seguendo i requisiti di una applicazione specifica. Esistono tool che permettono di selezionare i vari componenti e creano il software completo.
  - Esempio: **Hyperflex di Brugali**
    - ▷ il tool mostra graficamente un albero da cui si selezionano i vari componenti, si controlla che tutti i vincoli siano verificati e si preme un pulsante per generare il software.

3.3.1 Challenges - Tecniche di creazione di una SPL

La sfida in questione è quella di poter implementare soluzioni stabili attraverso l'uso delle SPL. Per creare una di queste esitono 3 diversi metodi:

- Proactive approach
  - Si crea una SPL che supporta tutti i prodotti che si vogliono.
- Extractive approach
  - modifica i sistemi pre-esistenti che hanno tante cose simili.
- Reactive approach
  - Modifcare una SPL esistente per far fronte a grosse e impreviste variazioni.

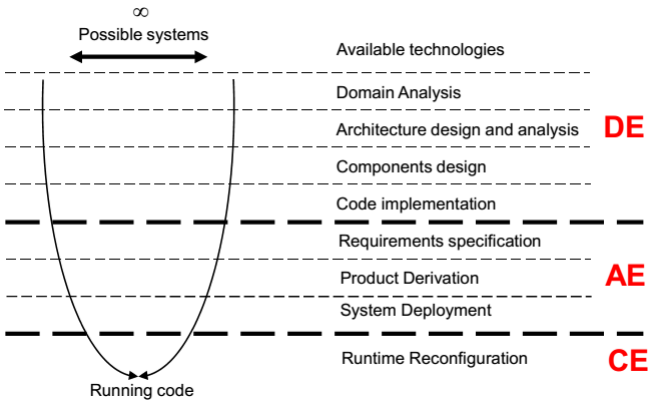


Figura 14: Advanced Software Product Line approach.

Nella parte finale della figura 14 (Runtime reconfiguration) si definisce la sfida dell'ingegneria del software applicata alla robotica, che permette al robot di riconfigurarsi in modo automatico in base a quelle che sono le esigenze dell'ambiente in cui opera (e.g. una volta attraversata un'area tortuosa con delle *gambe*, rimuovere le gambe e introdurre delle *ruote* per spostarsi).

4 Cinematica dei robot mobili su ruote

Locomotion il processo che permette a un robot di muoversi (forze (motori) da applicare sul carico (robot)).

Cinematica lo studio della matematica del moto, senza considerare le forze che agiscono sul moto. *Relazioni geometriche* che governano il sistema, e *relazioni* tra i parametri di controllo e il comportamento del sistema.

Dinamica studio del moto del robot in cui vengono modellate anche le forze (relazione tra moto e forze).

4.1 Cinematica dei robot mobili

Descrive il comportamento meccanico del robot per poterlo successivamente implementare e controllare (il controllore vuole sempre sapere come è fatto il robot).

Il robot si muove in un ambiente, ma non è possibile sapere dove si trova perchè non c'è un modo diretto per misurare la posizione del robot.

Quindi bisogna studiare la cinematica in modo da poter integrare nel tempo la posizione del robot, partendo da quella di partenza.

Questo processo porta all'aumento progressivo dell'errore (*problema principale dei robot mobili*).
Per capire come si muove il robot, bisogna capire come sono le sue ruote e come sono montate (vincoli delle ruote). Le ruote **non** sono **perfette** hanno anche movimenti lungo gli altri due assi.

IPOTESI

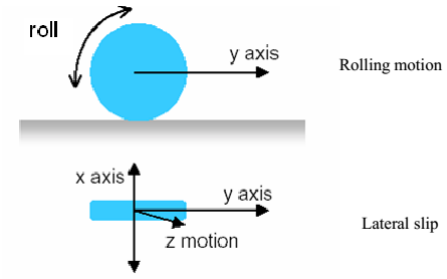


Figura 15: Modello ideale della ruota.

- la ruota ha al masimo un link di sterzatura, avente asse fisso al robot;
- la ruota è disposte in modo $\perp$ al pavimento;
- la ruota è fissata nella struttura del robot e non si muove;
- non c'è slip della ruota;
- la ruota non trasla e quindi si sposta solo lungo l'asse x;

Se queste assunzioni in genere funzionano bene e non conviene complicare il modello perchè l'errore di misura sulla velocità è quello prevalente.

Per le basse velocità a cui si muovono i robot, il modello della ruota modellato dal rotolamento va benissimo.

$$\begin{cases} r & raggio\,della\,ruota \\ v & velocità\,lineare\,della\,ruota \\ \omega & velocità\,angolare \\ t & velocità\,di\,sterzatura \end{cases}$$

inoltre si ha che

$$r = \frac{v}{\omega}$$

4.1.1 Tipologie di ruote

- **fissa**: gira solo rispetto al suo centro e non può sterzare (BART).
- **a centro orientabile**: gira rispetto al suo centro e inoltre può ruotare attorno a un asse verticale.
- **a centro spostato**: l'asse dello sterzo non interseca l'asse di rotazione, inoltre l'asse dello sterzo si trova a una distanza d. La forza applicata sull'asse verticale genera una coppia d che permette alla ruota di sterzare (il body trascina la ruota per il punto a cui è agganciata).
- **svedese**: la ruota è fissa sul body del robot, ma ha dei rulli, liberi di ruotare al posto dello pneumatico, posti inclinati rispetto all'asse della ruota. I rulli lasciano un DOF allo «pneumatico»: movimento per rotollamento, movimento strisciando (si muove latelarmente il robot, facendo girare le ruote opportunamente).

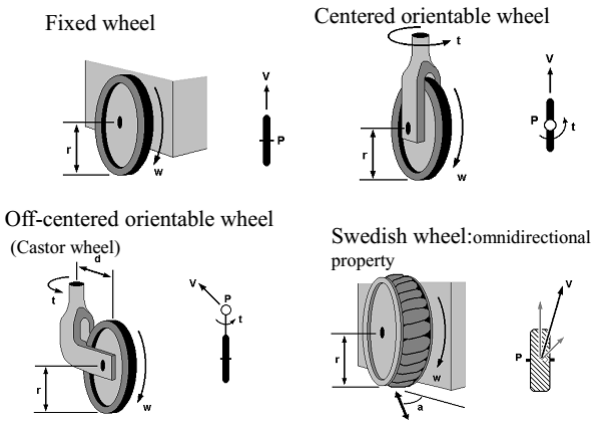


Figura 16: Tipologie di ruote.

4.1.2 Tipologie di robot

- **due ruote (*differential drive*)**:
  - ha un movimento fluido, la superficie di contatto della ruota si approssima a un punto;
  - può slittare quando sterza, dato che la superficie di contatto è piccola;
  - disegna degli archi di circonferenza quando curva attorno a un punto.
- **cingolato**:
  - grazie ai cingoli ha una superificie di contatto molto maggiore di quella del differential drive.
  - slitta di mento quando sterza perchè ha più aderenza.
  - si muove in base a quale cingolo fa contatto (caso di un cingolo in una buca e l'altro no).
  - difficile da modellare la virata, dato che il cingolo tocca il terreno in più punti.
- **omnidirezionale**
  - la velocità di avanzamento non è proporzionale a quella di rotazione;
  - quando due ruote si muovono, la terza striscia;
  - introducendo dei rulli come pneumatici, lo strisciamento si riduce grazie al loro rotolamento.

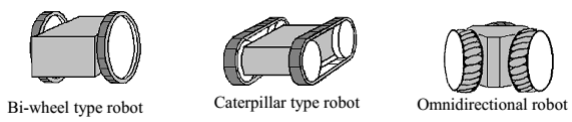


Figura 17: Tipologie di robot su ruote.

4.2 Centri di curvatura e rotazione

Tracciando l'asse di ogni ruota del robot quando curva, possiamo notare che si incontrano in un punto detto *ICR* (*Instantaneous Center of Rotation*) oppure *ICC* (*Instantaneous Center of Curvature*).

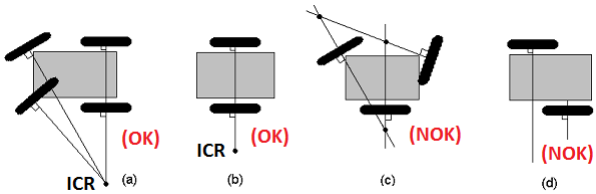


Figura 18: Centri di istantanea rotazione.

Dalla figura 18 si denota

1. Nel caso dei robot in (a) e in (b), le ruote sono posizionate in modo tale da conoscere a priori l'ICR.
   (a) caso dell'automobile: le ruote anteriori non sterzano dello stesso angolo perchè altrimenti le ruote posteriori strisciano e non si muovono per rotolamento.
   (b) ICR dipende dalle velocità delle ruote:
      i. $(v_{destra}, v_{sinistra}) = (0, k) \longrightarrow$ICR è sulla ruota destra.
      ii. $(v_{destra}, v_{sinistra}) = (k, k) \longrightarrow$ICR è all'infinito (va diritto).
      iii. $(v_{destra}, v_{sinistra}) = (k, -k) \longrightarrow$ICR è nel centro del robot.
2. Nel caso dei robot in (c) e in (d), le ruote non sono posizionate in modo da avere un ICR noto a priori (robot non ben definito).

Vincoli non olomonici sono vincoli sulle velocità del body del robot.

Caso del robot PIONEER:

- ☺ il robot può muoversi avanti e indietro istantaneamente
- ☹ il robot può muoversi di lato istantaneamente (deve fare una serie di pasaggi)
- le velocità dipendono da alcuni fattori: la struttura, ecc.

4.2.1 Notazione

Si definiscono alcune nozioni circa la matematica applicata alla geometria dei robot.

- $\{X_m, Y_m\}$: frame mobile;
- $\{X_b, Y_b\}$: frame di base;
- $q = [x\, y\, \theta]'$: posizione del robot nel frame di base;
- **matrice di rotazione**: rappresenta l'orientamento del frame di base rispetto al frame mobile

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \cos(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

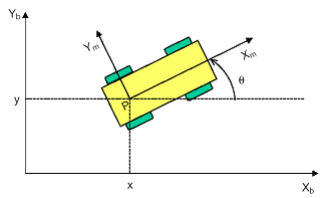


Figura 19: Robot Frames.

4.2.2 Differential drive

Si definiscono

- $Twist\,(v, \omega)$: v è la velocità lineare e ω la velocità di rotazione (VALORI DI CONTROLLO).
- R : raggio di curvatura
- $v = R \cdot \omega$

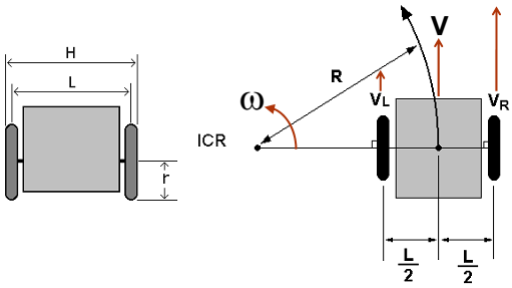


Figura 20: Differential drive frames.

Relazione di cinematica diretta se la ruota destra gira con velocità v_R e quella sinistra con v_L allora

$$\begin{aligned} v &= \frac{v_R + v_L}{2} \\ \omega &= \frac{v_R - v_L}{L} \end{aligned}$$

dove L è la distanza tra le due ruote, quindi si ottiene il raggio di rotazione R come

$$R = \frac{v}{\omega} = \frac{L}{2} \cdot \frac{v_R + v_L}{v_R - v_L}$$

Relazione di cinematica inversa invertendo le relazioni dirette si ottiene

$$\begin{aligned} v_L &= \omega \cdot \left(R - \frac{L}{2}\right) \\ v_R &= \omega \cdot \left(R + \frac{L}{2}\right) \end{aligned}$$

4.2.3 Triciclo

Il mezzo ha una ruota anteriore che può sterzare e può ruotare grazie a un motore. Le ruote posteriori invece ruotano liberamente.

Per controllare il triciclo si usanto i seguenti parametri

- α angolo di sterzatura.
- ω velocità di rotazione.

Inoltre sono note anche le seguenti dimensioni

- r raggio della ruota motrice.
- d distanza dal centro della ruota anteriore all'asse delle ruote posteriori.

L'**ICR** è **sempre** sull'asse delle ruote posteriori, come mostrato in figura 21, e si ottiene trovando l'intersezione tra l'asse delle ruote posteriori e la retta con angolo α che passa per il centro della ruota motrice.

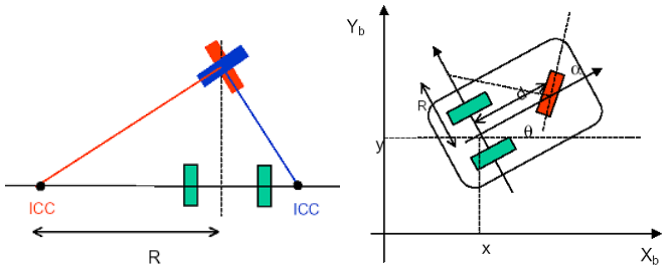


Figura 21: Triciclo.

Quindi il raggio di curvatura del robot attorno all'ICR è

$$R = d \cdot \tan\left(\frac{\pi}{2} - \alpha\right)$$

mentre la velocità lineare è data dalla ruota motrice e dalla sua direzione

$$v = \omega \cdot r \cdot \cos(\alpha)$$

Se si considera come centro del robot la ruota anteriore allora il raggio di curvatura diventa:

$$\sqrt{d^2 + R^2}$$

allora la velocità di rotazione la si può scrivere come

$$\omega\,(t) = \frac{v}{\sqrt{d^2 + R(t)^2}} = \frac{v}{d} \cdot \sin(\alpha)$$

da cui si ottengono le formule odometriche

$$\begin{aligned} \dot{x} &= \omega \cdot r \cdot \cos\,(\alpha)\cos(\theta) \\ \dot{y} &= \omega \cdot r \cdot \cos\,(\alpha)\sin(\theta) \\ \dot{\theta} &= \frac{v}{d} \cdot \sin\,(\alpha) \end{aligned}$$

Queste formule vanno poi discrettizzate per poterle usare sul controllore.

4.2.4 Synchronous drive

Ogni ruota è in grado di sterzare, quindi variando continuamente la direzione delle ruote il robot si muove diritto, allora l'**ICR** è **sempre** all'**infinito** (IN OGNI CASO).

Al variare dell'orientamento delle ruote cambia anche la direzione dell'ICR. Le ruote si muovono tutte assieme verso la stessa direzione a cui si vuole l'ICR.
Tutte le ***ruote*** sono ***sempre parallele*** e l'orientamento del robot avviene controllando direttamente i motori.

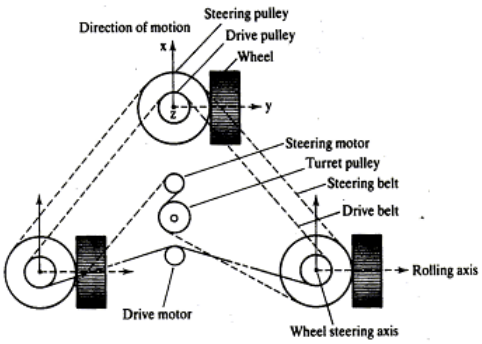


Figura 22: Synchronous drive.

4.2.5 Omnidirezionale

Dato che ci sono 3 parametri (x, y, θ), servono 3 motori per variarli in modo indipendente.
Se ha 2 motori il robot si dice ***sotto-attuato***, non è in grado di imporre una variazione di v istantanea.
Altrimenti se ha 4 motori, il robot è ***sovra-attuato***, quindi bisogna fare attenzione al coordinamento delle ruote per avere una variazione di v istantanea.

Il suo **ICR** è **sempre** nel **centro**.

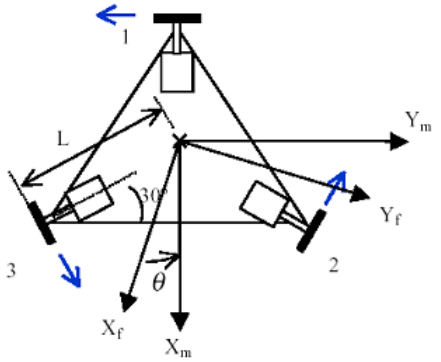


Figura 23: Robot omnidirezionale.

4.2.6 Automobile (Ackerman Steering)

La ruota frontale interna ruota leggermente di più rispetto a quella più esterna, per ridurre lo slittamento dello pneumatico delle ruote posteriori. L'Ackerman steering offre una soluzione abbastanza precisa per il dead-reckoning, supportando la trazione e la distanza da terra.

- d: distanza tra le ruote posteriori
- l distanza longitudinale tra le ruote
- θ_i angolo di sterzatura della ruota interna
- θ_o angolo di sterzatura della ruota esterna
- R distanza del ICR dal centro della distanza posteriore delle ruote

$$
\begin{aligned}
\cot(\theta) &= \frac{\cos(\theta)}{\sin(\theta)} \\
\cot(\theta_i) - \cot(\theta_o) &= \frac{R + \frac{d}{2}}{l} - \frac{R - \frac{d}{2}}{l} = \frac{d}{l}
\end{aligned}
$$

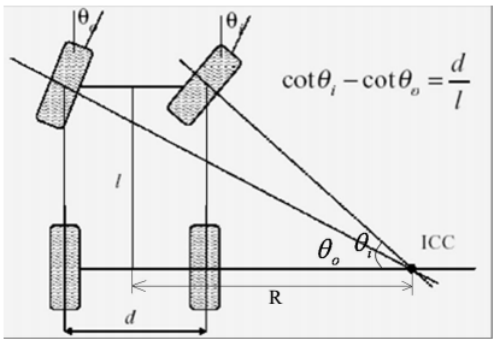


Figura 24: Ackerman steering.

Per questo motivo per controllare il rover si usa l'ackerman steering che sfrutta questa relazione per poter controllare le due ruote frontali come se fosse una sola ruota (volante macchina).

4.2.7 Carrello

Si suppone che il carrello sia trainato da una ruota virtuale con velocità v_1, quindi diventa come un triciclo con una ruota anteriore (quella virtuale).

$$\begin{aligned} \omega_1 &= \frac{v_1}{R_1} \qquad R_1 = |ICR_1 - P_1| \\ \omega_2 &= \frac{v_2}{R_2} \qquad R_2 = |ICR_2 - P_2| \end{aligned}$$

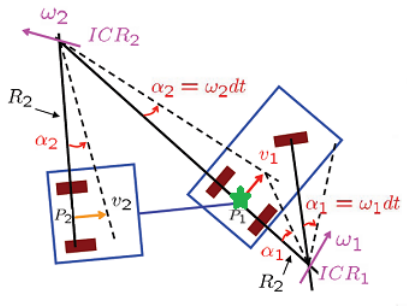


Figura 25: Robot articolato.

4.3 Odometria del differential drive

Si calcola la posizione del robot a partire dalle velocità delle ruote. Dato che il calcolo dell'odometria è incrementale e la velocità delle ruote è una stima (non quella esatta), allora c'è un errore (dovuto alle stime), che continua a crescere.

- ☺ va bene usare l'odometria per piccoli spostamenti del robot (quando l'integrazione dell'errore della posizione è ancora piccola).
- ☹ non va bene usarla per spostamenti lunghi.

In generale l'odometria è un'informazione che si riceve gratuitamente senza nessuna complicazione eccessiva (sensori encoder della ruota).

- D distanza dell'arco
- R raggio dell'arco
- φ angolo di rotazione
- r raggio delle ruota
- L distanza tra le ruote

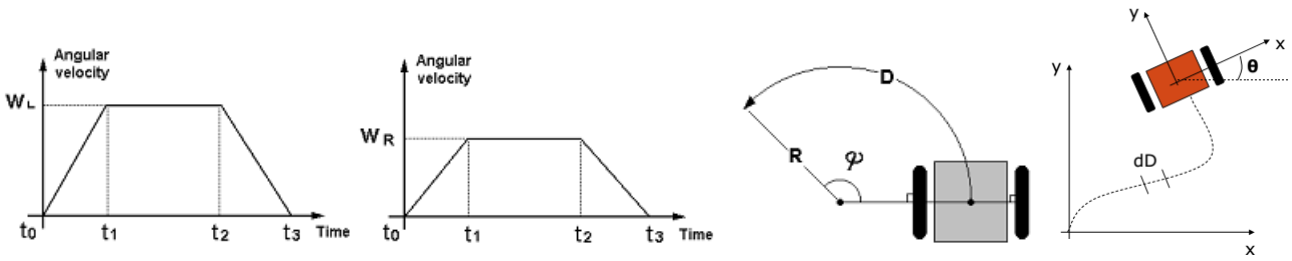


Figura 26: Rampe: profili delle velocità.

Date le velocità delle due ruote v_R e v_L si ha che

$$
\begin{aligned}
v(t) &= \frac{v_R(t)+v_L(t)}{2} = \frac{r}{2}\cdot(\omega_L(t)+\omega_R(t)) \\
\omega(t) &= \frac{v_R(t)-v_L(t)}{L} = \frac{r}{L}\cdot(\omega_R(t)-\omega_L(t))
\end{aligned}
$$

da cui si ottiene

$$
\begin{aligned}
D &= \int v(t)dt = \frac{r}{2}\cdot\frac{\omega_L+\omega_R}{2}(t_3-t_0+t_2-t_1) \\
\varphi &= \frac{D}{R} = \frac{r}{2\cdot L}(\omega_L+\omega_R)\cdot(t_3-t_0+t_2-t_1)
\end{aligned}
$$

quindi si ha

$$
\begin{aligned}
\dot{x} &= v(t)\cdot\cos(\theta(t)) \\
\dot{y} &= v(t)\cdot\cos(\theta(t)) \\
\dot{\theta} &= \omega(t)
\end{aligned}
$$

Ma per ricavare la posizione bisogna integrare le velocità qui sopra

$$
\begin{aligned}
x(t) &= \int v(t)\cdot\cos(\theta(t))\,dt = \frac{r}{2}\cdot\int (v_L(t)+v_R(t))\cdot\cos(\theta(t))\,dt \\
y(t) &= \int v(t)\cdot\sin(\theta(t))\,dt = \frac{r}{2}\cdot\int (v_L(t)+v_R(t))\cdot\sin(\theta(t))\,dt \\
\theta(t) &= \int \omega(t)dt = \frac{r}{L}\int (v_R(t)-v_L(t))\,dt
\end{aligned}
$$

Considerando le velocità costanti

$$
\theta(t) = \frac{1}{L}\cdot(v_R-v_L)\cdot t
$$

si ha

$$
\begin{aligned}
x(t) &= \frac{1}{2}\cdot(v_R+v_L)\cdot\int\cos\left(\frac{1}{L}\cdot(v_R-v_L)\cdot t\right)dt = \frac{L}{2}\cdot\frac{v_R+v_L}{v_R-v_L}\cdot\sin(\theta(t)) \\
y(t) &= \frac{1}{2}\cdot(v_R+v_L)\cdot\int\sin\left(\frac{1}{L}\cdot(v_R-v_L)\cdot t\right)dt = -\frac{L}{2}\cdot\frac{v_R+v_L}{v_R-v_L}\cdot\cos(\theta(t))
\end{aligned}
$$

ma sapendo che il raggio di curvatura è

$$
R = \frac{L}{2}\cdot\frac{v_R+v_L}{v_R-v_L}
$$

si scrive

$$
\begin{aligned}
x(t) &= R\cdot\sin(\theta(t)) \\
y(t) &= -R\cdot\cos(\theta(t)) \\
\theta(t) &= \frac{1}{L}\cdot(v_R-v_L)\cdot t
\end{aligned}
$$

Dato che θ è una funzione linerare si può notare che x e y sono funzioni periodiche, e infatti rappresentano una curva di circonferenza R.

Per poter implementare tali funzioni su un calcolatore bisogna discretizzarle. Quindi scelti il passo di discretizzazione Δt e il tempo iniziale t_0 si definisce la variabile tempo discreta $t_k = t_0 + k \cdot \Delta t$ ottenendo:

$$\begin{aligned} x(t_k) &= R \cdot \sin(\theta(t_k)) \\ y(t_k) &= -R \cdot \cos(\theta(t_k)) \\ \theta(t_k) &= \frac{1}{L} \cdot (v_R - v_L) \cdot t_k \end{aligned}$$

da cui si può ricavare che:

$$\begin{aligned} x(t_k) - x(t_{k-1}) &= R \cdot \sin(\theta(t_k)) - R \cdot \sin(\theta(t_{k-1})) \\ y(t_k) - y(t_{k-1}) &= -R \cdot \cos(\theta(t_k)) + R \cdot \cos(\theta(t_{k-1})) \\ \theta(t_k) - \theta(t_{k-1}) &= \frac{1}{L} \cdot (v_R - v_L) \cdot t_k - \frac{1}{L} \cdot (v_R - v_L) \cdot t_{k-1} \end{aligned}$$

e quindi:

$$\begin{aligned} x(t_k) - x(t_{k-1}) &= R \cdot [\sin(\theta(t_k)) - \sin(\theta(t_{k-1}))] \\ y(t_k) - y(t_{k-1}) &= -R \cdot [\cos(\theta(t_k)) - \cos(\theta(t_{k-1}))] \\ \theta(t_k) - \theta(t_{k-1}) &= \frac{1}{L} \cdot (v_R - v_L) \cdot \Delta t \end{aligned}$$

che è la variazione di posizione del rover nel tempo Δt andando a velocità costanti e quindi supponendo di conoscere un'approssimazione x_k della posizione del robot al tempo k si ottiene:

$$\begin{aligned} x_{k+1} &= x_k + R_k \cdot (\sin\theta_{k+1} - \sin\theta_k) \\ y_{k+1} &= y_k - R_k \cdot (\cos\theta_{k+1} - \cos\theta_k) \\ \theta_{k+1} &= \theta_k + \frac{1}{L} \cdot (v_R - v_L) \cdot \Delta t \end{aligned}$$

dove R_k è il raggio di curvatura nell'istante k. Questo si può scrivere meglio definendo:

$$\begin{aligned} SR_k &= v_{R,k} \cdot \Delta t \\ SL_k &= v_{L,k} \cdot \Delta t \end{aligned}$$

ossia lo spazio percorso dalle due ruote nel tempo Δt e notando che:

$$\begin{aligned} R_k &= \frac{L}{2} \cdot \frac{v_{R.k} + v_{L.k}}{v_{R,k} - v_{L,k}} \\ &= \frac{L}{2} \cdot \frac{v_{R.k} + v_{L.k}}{v_{R,k} - v_{L,k}} \cdot \frac{\Delta t}{\Delta t} \\ &= \frac{L}{2} \cdot \frac{v_{R.k} \cdot \Delta t + v_{L.k} \cdot \Delta t}{v_{R,k} \cdot \Delta t - v_{L,k} \cdot \Delta t} \\ &= \frac{L}{2} \cdot \frac{SR_k + SL_k}{SR_k - SL_k} \end{aligned}$$

si ottiene:

$$\begin{aligned} x_{k+1} &= x_k + R_k \cdot (\sin\theta_k - \sin\theta_{k-1}) \\ y_{k+1} &= y_k - R_k \cdot (\cos\theta_k - \cos\theta_{k-1}) \\ \theta_{k+1} &= \theta_k + \frac{SR_k - SL_k}{L} \end{aligned}$$

4.3.1 Effector noise

Dato che l'odometria si basa solo sui sensori e la posizione si ottiene integrando, questo continua ad aumentare nel tempo per questo motivo è solitamente affiancata da altri sensori ottenendo la *dead reckoning*.

Solitamente si usa l'odometria per brevi percorsi e poi si aggiorna la posizione usando un sensore con maggiore precisione.
L'odometria presenta anche degli errori di calibrazione che sono deterministici e quindi si possono compensare facilmente.

In generale gli errori sulla rotazione sono maggiori di quelli sulla distanza.
Il movimento del robot è rilevato dagli encoder e/o dai sensori di direzione (ad es. il giroscopio). Un maggiore numero di sensori aiuta a ridurre l'errore di posizionamento dovuto alla loro imprecisione o a causa esterne (ad es. lo slittamento). Gli errori si accumulano nel corso dei movimenti.

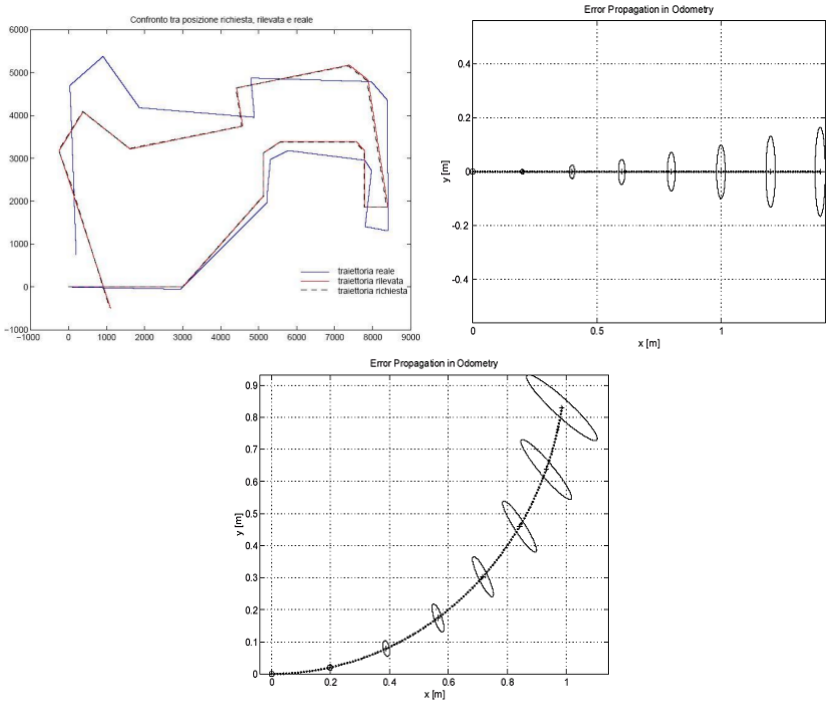


Figura 27: Profili errore nell'odometria.

Esempio

- Usare gli encoder su ruote non motrici (del carrello) è meglio perchè se il robot è fermo è fermo anche l'encoder e quindi se quelle motrici slittano queste si fermano e quindi riducono l'errore dell'odometria.

5 Controllo del moto

Obiettivo far seguire al robot una traiettoria, che è definita come una sequenza di punti (*waypoint*) che sono connessi diversamente tra di loro:

- **Rettilinei** in cui definiscono in modo univoco l'angolazione di arrivo;
- **Circolari** anche loro sono definiti univocamente come la retta, ma con un angolo di arrivo differente.

Per definire una traiettoria vengono usate le spline, per avere sempre curve senza spigoli e discontinuità.

Problema Discretizzando il percorso in una serie di elementi geometrici elementari (rette e archi di circonefernza), il controllore non considera le dinamiche del sistema.

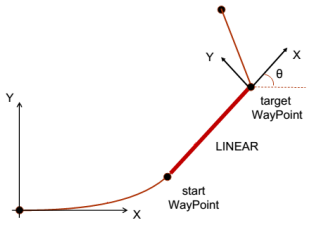


Figura 28: Traiettoria da percorrere.

Legge oraria Per eseguire movimenti lungo i waypoint, è meglio utilizzare una legge oraria.

Ossia il robot parte da una velocità iniziale stabilita e accellera fino a raggiungere la velocità massima, o di crociera. Una volta avvicinatosi alla fine inizia a decellerare per raggiungere la velocità finale.

Questo sistema minimizza il tempo per arrivare al prossimo waypoint, rispettando i limiti di accelerazione e decelerazione del robot e garantendo le velocità iniziali e finali volute.

Errore sulla traiettoria - trajectory interpolator Il robot non è in grado di seguire la traiettoria esattamente, perchè non riesce a fare tutti i movimenti possibili e inoltre la sua risposta non è sempre uguale a quella desiderata (da noi o dal controllore).

Per alleviare questo problema si divide il percorso in tanti waypoint molto vicini tra loro, in modo che il robot se devia dal percorso cercarà di riportarsi sulla traiettoria per raggiungere il prossimo waypoint (al prossimo punto corregge la direzione).

Pertanto se i waypoint sono abbastanza vicini tra di loro, il robot segue esattamente il percorso definito.

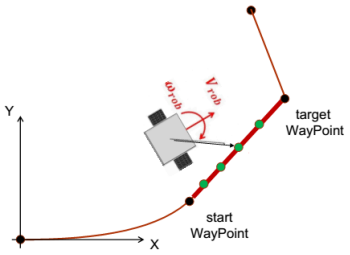


Figura 29: Interpolazione della traiettoria.

Per ogni tipo di segmento di una traiettoria è necessario implementare uno specifico interpolatore.

5.1 Implementazione del controllo di moto

Per implementare il controllo di moto, servono 3 elementi fondamentali.

5.1.1 Trajectory follower

Il robot usa la ***leggere oraria***, quindi dalla posizione corrente in cui si trova e dalla traiettoria, calcola il prossimo punto che deve raggiungere dividendo il percorso in una serie di waypoint come visto sopra.

La traiettoria può essere percosa in due modi:

- arrivando nei punti di controllo ad un istante temporale predefinito
- arrivando con una velocità specifica, particolare per ogni segmento

5.1.2 Pose controller

Riceve in input il punto finale da raggiungere, e fornisce al robot i comandi di twist necessari per raggiungerl.

Il pose controller è diverso per ogni tipo di robot, perchè in base al robot utilizzato, cambiano anche i comandi di twist.
Ci sono diversi modi per fare un pose control:

- usando un percorso a cubica.
- usando il kinematic pose control

5.1.3 Ramp generator

Il ramp generato implementa la legge orario, così da calcolare la velocità finale in funzione della distanza dal prossimo waypoint, per poi darla in input al pose controller.

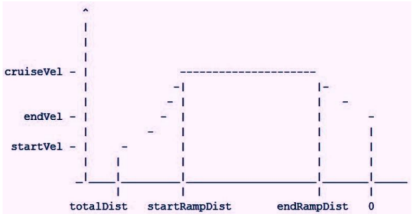


Figura 30: Generazione della rampa di velocità.

5.2 Recovery trajectory

Se il robot non si trova sulla traiettoria, deve riprenderla con la giusta inclinazione per poterla seguire.

Bisogna quindi usare una curva che permette di regolare anche l'inclinazione di arrivo e quindi si utilizza una cubica, dato che servono 4 parametri (valori iniziali e finali e la loro derivata che dipende dall'inclinazione)

$$y = a + b \cdot x + c \cdot x^2 + d \cdot x^3$$

La cubica viene ricalcolata ad ogni periodo.

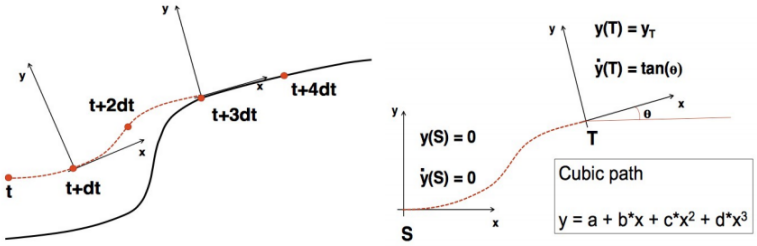


Figura 31: Traiettoria di recupero.

Bisogna però trovare un modo per scegliere il punto del percorso migliore per raggiungerlo e solitamente si sceglie quello che minimizza la lunghezza dell percorso da fare (salvo ostacoli).

5.3 Kinematic pose

Sfrutta un controllore proporzionale per portare la posizione iniziale più simile a quella finale.

Se si usa questo sistema bisogna garantire che il controllore sia stabile e che quindi non diverga.
Sia dato il robot differential drive in figura 32, la sua cinematica nel frame iniziale $\{x_I, y_I, q\}$ è data da

$$_I \begin{bmatrix} \dot{x} \\ \dot{y} \\ \theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix}$$

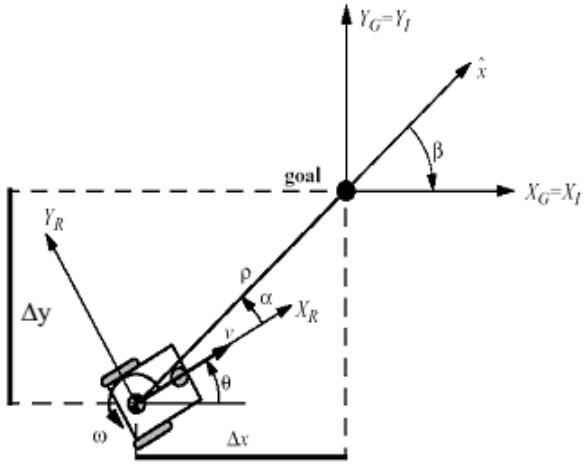


Figura 32: Kinematic pose di un differential drive.

dove α denota l'angolo tra l'asse x_R del punto di riferimento del robot e il vettore che connette il centro dell'asse delle ruote con la posizione finale.

La trasformazione delle coordinate in coordinate polari con origine la posizione del goal:

$$\begin{aligned} \rho &= \sqrt{\Delta_x^2 + \Delta_y^2} \\ \alpha &= -\theta + \arctan^2(\Delta_x, \Delta_y) \\ \beta &= -\theta - \alpha \end{aligned}$$

Il sistema, descritto nelle nuove coordinate polari, si presenta come:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos(\alpha) & 0 \\ \frac{\sin(\alpha)}{\rho} & -1 \\ -\frac{\sin(\alpha)}{\rho} & 0 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$for \qquad \alpha \in I_1 \;\; I_1 = (-\frac{\pi}{2}, \frac{\pi}{2}]$$

$$forward\, direction$$

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & 0 \\ -\frac{\sin(\alpha)}{\rho} & 1 \\ \frac{\sin(\alpha)}{\rho} & 0 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$for \qquad \alpha \in I_2 \;\; I_2 = (-\pi, -\frac{\pi}{2}] \cup (\frac{\pi}{2}, \pi]$$

$$backward\, direction$$

Si può mostrare che con

$$\begin{aligned} v &= k_\rho \cdot \rho \\ \omega &= k_\alpha \cdot \alpha + k_\beta \cdot \beta \end{aligned}$$

che il sistema di controllo reatroazionato

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \cdot \rho \cdot \cos(\alpha) \\ k_\rho \cdot \sin(\alpha) - k_\alpha \cdot \alpha - k_\beta \cdot \beta \\ -k_\rho \cdot \sin(\alpha) \end{bmatrix}$$

può portare il robot in $(\rho, \alpha, \beta) = (0, 0, 0)$
Il segnale di controllo v ha sempre il segno costante, la direzione del movimento è mantenuta positiva (negativa) durante il movimento del robot.

Si può mostrare inoltre il sistema di controllo retroazionate è localmente esponenzialmente stabile se

$$\begin{aligned} k_\rho &> 0 \\ k_\beta &< 0 \\ k_\alpha - k_\rho &> 0 \end{aligned}$$

6 Obstacle avoidance

L'obiettivo è quello di evitare collisioni con gli ostacoli sul percorso, per raggiungere la destinazione, sfruttando una mappa locale del robot o delle informazioni date dai sensori.

Un obstacle avoidance efficiente può essere ottimale rispetto:

- all'obiettivo complessivo
- alla velocità attuale e alla cinematica del robot
- rispetto ai sensori "a bordo"
- rispetto al rischio attuale e futuro di collisione.

Esistono diversi algoritmi che implementano l'obstacle avoidance.

6.1 Bug algorithm

Il robot costeggia l'ostacolo mantenendolo sempre sulla stesso lato (o sulla sinistra oppure sulla destra), fino a tornare sulla traiettoria iniziale.

Questo algoritmo non è sempre la cosa migliore da fare in quando a volte è meglio:

- prendere l'altra direzione (destra o sinistra)
- l'ostacolo è tipo concavo e non c'è bisogno di seguire tutto il bordo della conca.

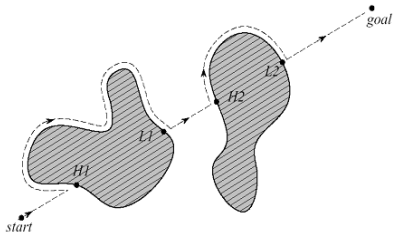


Figura 33: Bug algorithm.

6.2 Vector field histogram

L'ambiente viene rappresentato con una griglia e i valori delle celle della griglia rappresentano la probabilità che ci sia un ostacolo in quel determinato punto.

La griglia viene trasformata in un istogramma, di probabilità, e in n tutti gli istanti il robot calcola la distanza (statisticamente) del primo ostacolo che incontrerebbe andando in ogni direzione.

Quindi il robot è in grado di individuare tutte le aperture in cui può passare, viste come direzioni in cui la probabilità di trovare un ostacolo è inferiore a una soglia. Tutto ciò che si trova sopra la soglia è un ostacolo.

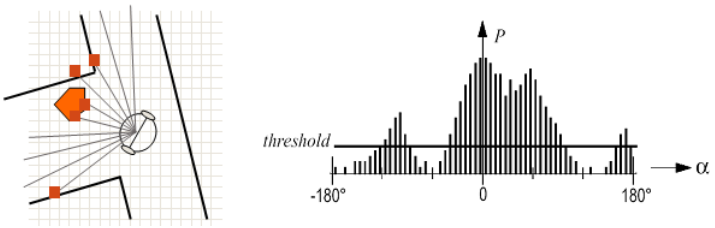


Figura 34: Bug algorithm.

Tra le varie aperture si sceglie quella che minimizza una funzione obbiettivo che valuta:

- la direzione che bisogna seguire per raggiungere il target
- l'orientamente delle ruota in questo istante
- la direzione scelta all'istante prima dando una importanza ai tre contributi calibrabile.

6.3 Basic Curvature Velocity Methods

Questo algoritmo consiste nell'aggiungere dei limiti ai twist che si possono dare al robot in base ai limiti fisici dell'ambiente.

Per esempio se il robot può fare solo curvature ci saranno dei limiti al raggio in modo da non prendere raggi che lo farebbero andare eventualmente contro un ostacolo.

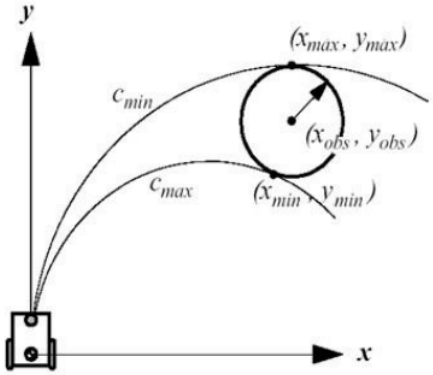


Figura 35: Basic curvature velocty method.

6.4 Trajectory rollout algorithm (TRA)

1. Prende dei campioni discreti dello spazio di controllo del robot $(dx, dy, d\theta)$
2. Per ogni campione di velocità esegue una proiezione per determinare dove si troverà il robot dopo un certo tempo a quella velocità e direzione
3. Calcola una traiettoria per ogni simulazione che tenga conto degli ostacoli, della destinazione e della velocità di transito.
4. Scarta le traiettorie che collidono con ostacoli
5. Sceglie la traiettoria migliore rispetto all'obiettivo e genera un movimento conseguente.
6. Riesegue l'algoritmo una volta raggiunta la nuova posizione raggiunta.

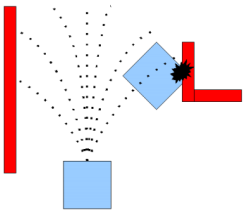


Figura 36: Trajectory rollout algorithm (TRA).

6.5 Dynamic window approach (DWA)

E' un algoritmo molto simile al TRA con la differenza che invece di campionare tutte le possibili velocità di avanzamento, considera solo le velocità che si possono raggiungere con un solo passo di simulzione.

Il TRA considera tutte le velocità ottenibili in tutto il piccolo periodo di tempo della simulazione e quindi sono molte di più e potrebbe avere prestazioni migliore a scapito della computazione, ma in realtà si è visto che le prestazioni sono paragonabili e quindi si usa il DWA di solito.

Per eseguire il DWA è necessaria una mappa che solitamente è a griglia in cui in ogni cella c'è indicata la distanza dall'obbiettivo (continuamente aggiornata) e se c'è un ostacolo.

Nella valutazione delle varie velocità del DWA si tiene conto anche se il cambio di cella sulla mappa comporta un abbassamento della distanza verso il target.

Questa aggiunta alla valutazione potrebbe portare il robot a oscillare tra due celle e quindi si rende invalida la direzione opposta a quella del passo prima.
La velocità ammissibile alla simulazione per il DWA sono quelle che permettono al robot di fermarsi prima di raggiungere l'ostacolo

$$Va = \left\{(v,\omega) \mid v \leq \sqrt{2 \cdot dist(v,\omega) \cdot a_{trans}} \;\; \wedge \;\; \omega \leq \sqrt{2 \cdot dist(v,\omega) \cdot a_{rot}}\right\}$$

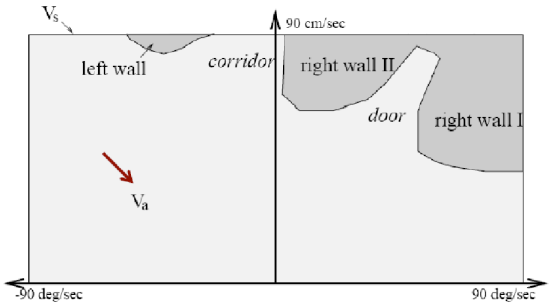


Figura 37: DWA: velocità ammissibili.

Le velocità invece raggiungibili con le accelerazioni sono

$$V_d = \{(v,\omega) \mid v \in [v - a_{trans} \cdot t, v + a_{trans} \cdot t] \;\; \wedge \;\; \omega \in [v - a_{rot} \cdot t, v + a_{rot} \cdot t]\}$$

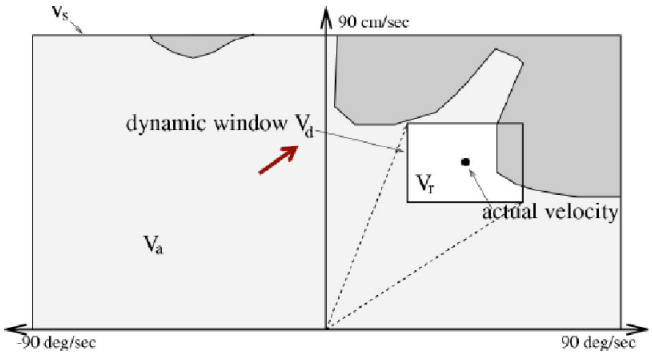


Figura 38: DWA: velocità raggiungibili.

Siano invece

- V_s tutte le velocità possibili, che il robot può raggiungere
- V_a area libera da ostacoli, per cui si hanno le velocità ammissibili
- V_d velocità raggiungibili dal robot, su sue possibili accelerazioni

Si ha lo spazio di ricerca (search space) definito come

$$V_r = V_s \cap V_a \cap V_d$$

6.5.1 Adattamento della traiettoria

Il DWA può essere integrato con il trajecotry follower infatti il robot mentre segue la direzione esegue anche il DWA, che entra in gioco quando il percorso passa su un ostacolo insuperabile.

1. Il percorso globale viene generato in anticipo
2. Il percorso è adattato se vengono incontrati degli ostacli
3. DWA considera anche l'ingombro che ha il robot

4. è real time perchè si valuta solo la massima velocità

La funzione obiettivo di selezione è

$$\max\left(a \cdot speed + b \cdot distance + c \cdot global_heading\right)$$

dove

$$\begin{aligned} speed &= \frac{v}{v_{max}} \\ distance &= \frac{L}{L_{max}} \\ global_heading &= 1 - \frac{\alpha - \omega \cdot T}{\pi} \end{aligned}$$

Map grid

Per calcolare le traiettorie ottimali si usano mappe a griglia su cui viene mappato il percorso. La griglia viene ricalcolata attorno al robot ad ogni ciclo di controllo.

In ogni quadratino si hanno due informazioni:

- **oscurato** contiene un ostacolo
- **numero** che indica la distanza dal percorso a quella dell'obiettivo (marcato con zero) e combinata.

L'algoritmo calcola il percorso migliore sfruttando la distanza di Manhattan (la distanza tra due punti è la somma del valore assoluto della differenza delle loro coordinate).

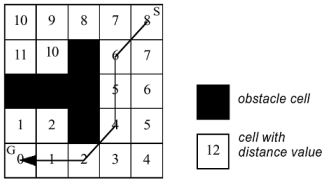


Figura 39: Griglia per l'adattamento della traiettoria con DWA.

Usando un approccio globale per l'applicazione del DWA per l'adattamento della traiettoria alla mappa in figura 39, si aggiunge una minima-free function NF1 (onda di propagazione) alla funzione obiettivo definita sopra.

Inoltre la griglia occupancy grid viene aggiornata con misure di distanza.

6.6 Oscillation Suppression

L'oscillazione di verifica quando in tutte e tre le dimensioni (x, y, θ) i valori positivi e negativi sono scelti consecutivamente.

Per evitarle, quando il robot si muove in qualche direzione, per il ciclo dopo la direzione opposta è marcata come non valida, fino a quando il robot si è mosso al di là di una certa distanza dalla posizione dopo il flag è stato settato.

7 Line following

Obbiettivo tramite un sensore (fotocamera o videocamera), il robot acquisice delle immagini così da rielaborarle per indivduare la linea che deve seguire.

7.1 Rielaborazione dell'immagine

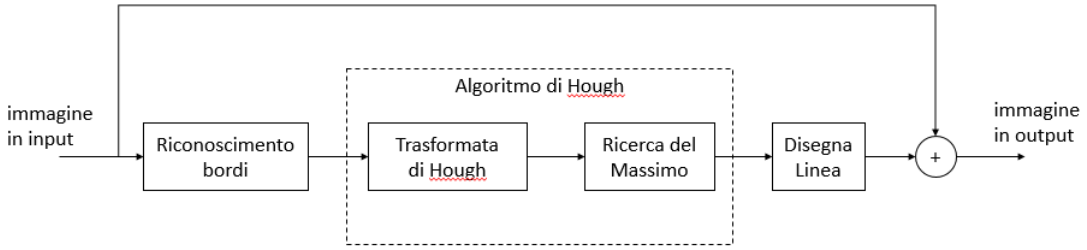


Figura 40: Ricerca delle linee nelle immagini: algoritmo di Hough.

Per rielaborare l'immagine si eseguono le operazioni descritte in figura 40:

- **Edge detection**
  - prende in input un'immagine e ne restituisce una nuova in cui sono messi in evidenza i bordi dei vari oggetti individuati.
- **Hough algorithm**
  - che individua tutte le linee mettendole in un *array accumulatore*, e da qui individua la più lunga (r_m, θ_m).
- **Draw line**
  - che restituisce una immagine con disegnata solo la linea scelta, definita dall'equazione

    $$r_m = x \cdot \cos(\theta_m) + y \cdot \sin(\theta_m)$$

    Tale linea è definita dai punti (x, y)che hanno «votato» per la linea più lunga (r_m, θ_m).

7.1.1 Image convolution

Filti lineari

Per modificare le immagini si usano dei kernel, cioè delle matrici quadrate con dimensioni dispari, che danno un peso ai pixel adiacenti e a quello centrale.

Il kernel viene applicato a tutti i pixel e quindi viene ricalcolato il colore di quel pixel come la somma pesata indicata sulla matrice.
Il kernel di solito è una matrice più piccola dell'immagine che deve modificare.

Per esempio:

- $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ copia l'immagine esattamente.
- $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ trasla l'immagine di un pixel
- $\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ tfa la media dei pixel adiacenti ottenendo l'effetto blur.
- Calcolo dell'immagine modificata.

$$\underset{\substack{immagine\\originale}}{\begin{bmatrix} 10 & 5 & 3 \\ 4 & 6 & 1 \\ 1 & 1 & 8 \end{bmatrix}} \longrightarrow \underset{kernel}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 1 & 0.5 \end{bmatrix}} \longrightarrow \underset{\substack{immagine\\modificata}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0 \end{bmatrix}}$$

$$\begin{aligned} 8 &= [(10*0)+(5*0)+(3*0)]+[(4*0)+(6*0.5)+(1*0)]+[(1*0)+(1*1)+(8*0.5)] \\ &= [0]+[3]+[1+4] \\ &= 3+5 \end{aligned}$$

Filtri derivativi

L'immagine viene rappresentata come una superficie, con altezze corrispondente ai valori di grigio. Mentre la luminosità è una fuzione che dipende da due variabili sul piano dell'immagine (valore del livello di grigio$z = f(x, y)$).

I contorni sono pixel in cui la funzione di luminosità cambia bruscamente come mostrato in figura 41. Quindi i cambiamenti possono essere descritti facendo le derivate parziali di z.

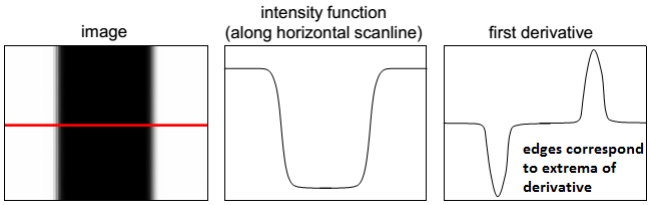


Figura 41: Bordi.

Il gradiente di un'immagine generica corrisponde a

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

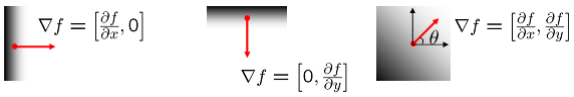


Figura 42: Gradienti di immagini.

Da qui si calcola la direzione del gradiente

$$\theta = tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)$$

e l'intensità del bordo

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

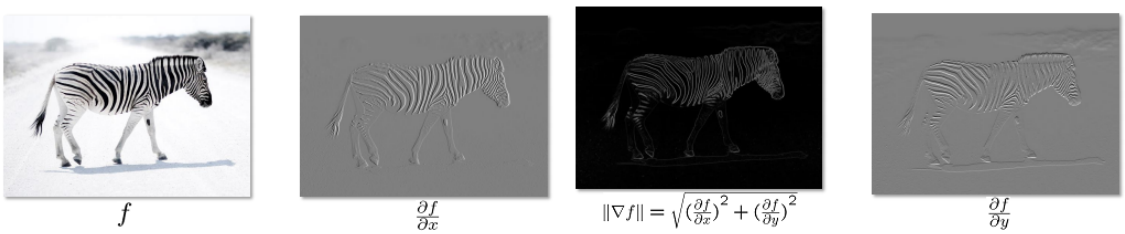


Figura 43: Gradienti di un immagine generica.

7.1.2 Riconoscimento dei bordi

I bordi sono aree dell'immagine con un alto contrasto, e di solito si usano per rappresentare i confini degli oggetti.

Quindi il riconoscimento dei bordi è utilizzato nella segmentazione dell'immagine, usata per suddividere l'immagine in aree corrispondenti a diversi oggetti.

In teoria, siccome i bordi si presentano in alta frequenza, potremmo riconoscerli applicando un filto passa alto (nel dominio di Fourier) ma questo sistema è molto inefficiente e si può fare di meglio.

In pratica, il riconoscimento dei bordi avviene nel *dominio dello spazio* perchè è meno oneroso in termini computazionali e di solito fornisce risultati migliori.

Tale procedimento consiste nello svolgimento dei seguenti punti

1. **Riduzione del rumore**: si cerca di ridurre il rumore nell'immagine il più possibile, evitando di togliere informazione anche ai bordi significativi.

2. **Miglioramento dei bordi**: valorizzazione dei contorni con dei filtri appositi che fanno risaltare i contorni e oscurano il resto.

3. **Riconoscimento dei bordi**: si scelgono tra tutti i valori in uscita al filtro quelli che sono effettivamente contorni, scartando il resto.

7.1.3 Trasformata di Hough sul piano

Siano definiti:

- $\pi = (m, q)$ il piano parametrico definito dai parametri m e q.
- $\sigma = (x, y)$ un generico piano definito da x e da y.

Ogni retta può essere definita tramite i parametri m e q, quindi dato un piano σ è possibile trovare tutte le rette su di esso attraverso tutte le possibili combinazione di m e q. Quindi il piano parametrico π è un piano i cui punti rapprensentano una retta del piano σ.

Dato che:

$$y = m \cdot x + q$$

si ottiene che

$$q = -m \cdot x + y$$

e quindi è possibile mappare tutte le rette del piano parametrico π tramite tutte le combinazione di x e y.

Pertanto ogni retta del piano π rappresenta un punto del piano σ.

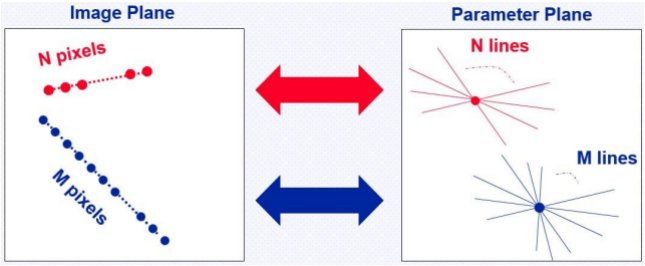


Figura 44: Trasformata di Hough sul piano.

Se si prendono N punti sulla retta $y = a \cdot x + b$ del piano σ e si disegnano le corrispettive N rette nel piano π queste passano perforza nel punto (a, b) perchè:

$$b = -a \cdot x + y$$

è verificata per tutti gli N punti.

Se si portano questi ragionamenti in un piano discreto, come un'immagine σ, il numero di pixel e di rette è limitato.

Allora è possibile assegnare a ogni punto del piano σ la lunghezza di tale retta, che corrisponde al numero di rette che si intersecano nel punto nel piano π.

7.1.4 Trasformata di Hough nello spazio

Le trasformata di Hough vista prima non permette di rappresentare le rette verticali perchè m dovrebbe essere infinito. Quindi si può sfruttare un secondo modo per rappresentare le rette, cioè quello con coordinate polari:

$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

e applicare gli stessi concetti al piano r, θ.
L'unica differenza è che nel piano r, θ le rette non hanno nessun significato dato che non è più possibile fare quella semplice trasformazione.

In compenso è possibile fare un analogo ragionamento con un polinomio trigonometrico infatti tutti i punti appartenenti a una retta r, θ nel piano x, y devono rispettare l'equazione:

$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

che corrisponde a un polinomio trigonometrico nel piano r, θ. Inoltre i punti nel piano x, y sono delle sinusoidi che si intersecano nel piano r, θ.

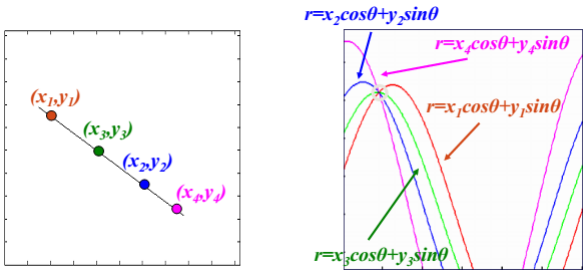
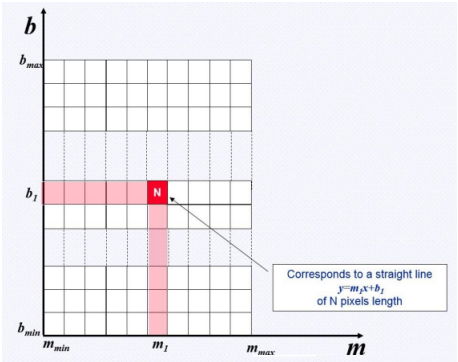


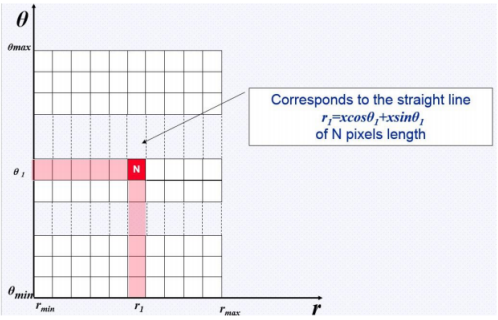
Figura 45: Accumulatore in r, θ.

7.1.5 Accumulatori

Gli accumulatori che genera l'algoritmo di Hough sono semplicemente delle matrici sul piano.



(a) Accumulatore per lo spazio m, q.



(b) Accumulatore per lo spazio r, θ.

Figura 46: Accumulatori dell'algotimo di Hough.

7.2 Visual Servoing for Line Following

Obiettivo Il robot, munito di una telecamera che gli fornisce una finestra di visibiltà (rappresentata in azzurro), deve seguire il percorso rosso p usando una legge di controllo ben definita.

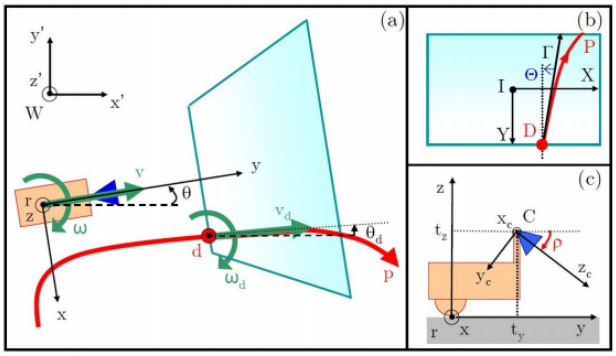


Figura 47: Esempio task da compiere.

Nello schema di path following in figura 47 (b) si utilizzano due caratteristiche del piano dell'immagine:

- la posizione del percorso (punto ai bordi dell'immagine);
- l'orientamento tangente al percorso.

Ipotizzando che una porzione del percorso sia sempre visibile, si utilizzano le caratteristiche descritte sopra per ottenere una proiezione del percorso sul piano dell'immagine $D = [X\,Y] \cdot T$.

Quindi si hanno

- P : la proiezione del percorso p sul piano dell'immagine
- Γ : la tangente di P orientata secondo la riezione del percorso D
- Φ : l'offset angolare tra Γ e l'asse $-Y$ (senso antiorario)

Legge di controllo

Il sensore di immagine restituisce un'immagine contenente una freccia il cui inizio rappresenta la posizione del robot e la linea della freccia rappresenta la linea da seguire.

In generale in robot cerca di spostare l'inizio della freccia in basso al centro della finestra di visibilità, e quindi cerca di tenere la linea sotto di lui ed esattamente al centro.

Per farlo il robot imposta una velocità costante v e poi tramite una legge di controllo non lineare retroazionata di ricava la ω.

In genere si divide il problema in due sottoproblemi:

- **column controller**: cerca di portare il pallino rosso in basso;
- **row controller**: cerca di portare il pallino al centro.

Prima si utilizza il controllo di colonna per poter rivolgere il robot verso la linea, solo successivamente interviene il row controller per portarlo sulla linea.

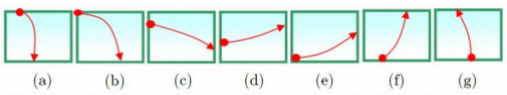


Fig. 2. Seven possible configurations of P in the image (D is represented by the red circle).

Figura 48: Esempio task da compiere 2.

8 Geometria 3D

Posizione assoluta i corpi si riferiscono tutti a un sistema di riferimento fisso

Posizione relativa i corpi non sono vincolati a un solo sistema di riferimento, posso averne quanti ne ho voglia

8.1 Geometria delle traslazioni

Semplice traslazione da un punto iniziale A a un punto finale B

$$
\begin{aligned}
\begin{bmatrix} qx_0 \\ qy_0 \\ qz_0 \end{bmatrix} &= \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} qx_1 \\ qy_1 \\ qz_1 \end{bmatrix} \\
\mathbf{Q}^1 &= \mathbf{T}_1^0 + \mathbf{Q}^0 \\
\mathbf{Q}^0 &= \mathbf{T}_0^1 + \mathbf{Q}^1 \\
\mathbf{T}_1^0 &= -\mathbf{T}_0^1 \quad (traslazione\, inversa)
\end{aligned}
$$

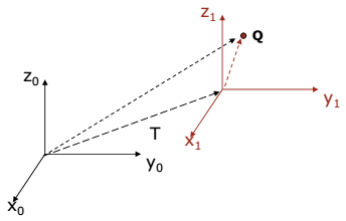


Figura 49: Traslazione.

8.2 Geometria delle rotazioni

Rotazione rispetto all'asse z.

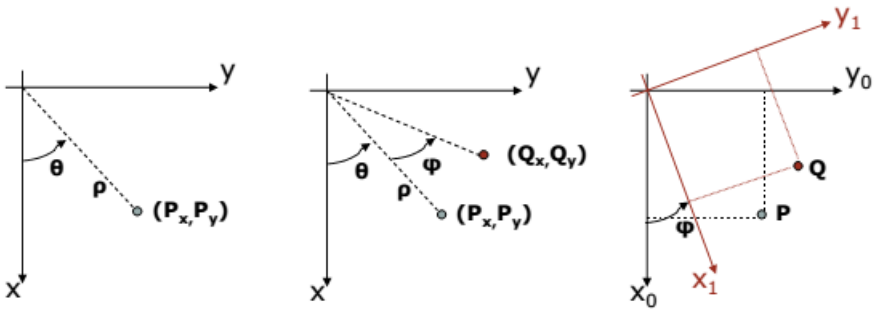


Figura 50: Rotazione attorno all'asse z.

$$
\begin{aligned}
coordinate\, polari \quad &\rightarrow \quad \begin{cases} P_x = \rho cos(\theta) \\ P_y = \rho sin(\theta) \end{cases} \\
\rho &= \sqrt{P_x^2 + P_y^2} \\
\theta &= arctan\left(\frac{P_y}{P_x}\right)
\end{aligned}
$$

$$
\begin{aligned}
Q_x &= \rho cos\,(\theta+\varphi) \quad Q_y = \rho sin\,(\theta+\varphi) \\
Q_x &= \rho cos\,(\theta)\,cos\,(\varphi) - \rho sin\,(\theta)\,sin\,(\varphi) = P_x cos\,(\varphi) - P_y sin\,(\varphi) \\
Q_y &= \rho cos\,(\theta)\,sin\,(\varphi) + \rho sin\,(\theta)\,cos\,(\varphi) = P_x sin\,(\varphi) + P_y cos\,(\varphi) \\
\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} &= \begin{bmatrix} cos\,(\varphi) & -sin\,(\varphi) \\ sin\,(\varphi) & cos\,(\varphi) \end{bmatrix} \cdot \begin{bmatrix} P_x \\ P_y \end{bmatrix} \\
\mathbf{Q} &= \mathbf{R} \cdot \mathbf{P} \\
\mathbf{Q}^1 &= \mathbf{P}^0 \\
\mathbf{Q}^0 &= \mathbf{R}_0^1 \cdot \mathbf{Q}^1 = \mathbf{R}_0^1 \cdot \mathbf{P}^0 \\
\mathbf{Q}^1 &= \mathbf{R}_1^0 \cdot \mathbf{Q^0}
\end{aligned}
$$

Fornisce l'orientamento di una terna di coordinate rispetto ad un'altra.

I vettori colonna sono i cos direttori degli assi della terna ruotata rispetto alla terna di riferimento. Partendo dalla posizione

$$\mathbf{P} = \mathbf{R}\mathbf{P}' \ , \ con\ \mathbf{R}\ matrice\ di\ rotazione$$

si trova

$$\mathbf{P}' = \mathbf{R}^{-1}\mathbf{P} = \mathbf{R}^T\mathbf{P}$$

$$\mathbf{R^T} = \mathbf{R}^{-1} \ (ortogonalità\ di\ \mathbf{R})$$

dove $\mathbf{R}$ rappresenta l'operatore che permette di ruotare un vettore di un angolo prefissato, attorno ad un generico asse di rotazione nello spazio.

Le rotazioni successive sono espresse come successione di rotazioni parziali e si ottengono come

$$\mathbf{R}_2^0 = \mathbf{R}_1^0 \cdot \mathbf{R}_2^1$$

moltiplicazione da sinistra a destra delle rotazioni precedenti prese nell'ordine in cui sono avvenute. Le rotazioni successive rispetto ad una terma fissa si ottengono come

$$\check{\mathbf{R}}_2^0 = \check{\mathbf{R}}_2^1 \cdot \mathbf{R}_1^0$$

moltiplicazione da destra verso sinistra delle singole matrici di rotazione nell'ordine delle rotazioni.

8.3 Angoli Roll - Pitch - Yaw (RPY)

Usati in aeronautica, indicano rollio (roll), beccheggio (pitch) e imbardata (yaw) di uno scafo.

I tre parametri indicano rotazioni definite rispetto ad una terna fissa solidale al baricentro dello scafo.

8.4 Angoli di Eulero I

- Si ruota la terna origine di un angolo φ intorno all'asse z: la rotazione è descritta dalla matrice di rotazione $\mathbf{R}_{z,\varphi}$

- Si ruota la terna di un angolo θ intorno all'asse x' corrente: la rotazione è descritta dalla matrice di rotazione $\mathbf{R}_{x',\theta}$

- Si ruota la terna dell'angolo ψ attorno all'asse z'' corrente: la rotazione è descritta dalla matrice di rotazione $\mathbf{R}_{z'',\psi}$

$$\mathbf{R}_0^1 = \mathbf{R}_{z,\varphi} \cdot \mathbf{R}_{x',\theta} \cdot \mathbf{R}_{z'',\psi}$$

8.5 Angoli di Eulero II

- Si ruota la terna origine di un angolo φ intorno all'asse z: la rotazione è descritta dalla matrice di rotazione $\mathbf{R}_{z,\varphi}$
- Si ruota la terna di un angolo θ intorno all'asse y' corrente: la rotazione è descritta dalla matrice di rotazione $\mathbf{R}_{y',\theta}$
- Si ruota la terna di un angolo ψ attorno all'asse z'' corrente: la rotazione è descritta dalla matrice di rotazione $\mathbf{R}_{z'',\psi}$

$$\mathbf{R}_0^1 = \mathbf{R}_{z,\varphi} \cdot \mathbf{R}_{y',\theta} \cdot \mathbf{R}_{z'',\psi}$$

8.6 Coordinate omogenee

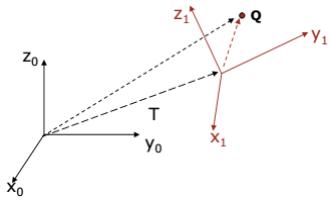


Figura 51: Roto-traslazione.

$$\begin{aligned}
\mathbf{Q}^0 &= \mathbf{R}_0^1 \cdot \mathbf{Q}^1 + \mathbf{T}_0^1 \\
\mathbf{Q}^0 &= \mathbf{H}_0^1 \cdot \mathbf{Q}^1
\end{aligned}$$

$$\begin{bmatrix} qx_0 \\ qy_0 \\ qz_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_0^1 & \mathbf{T}_0^1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} qx_1 \\ qy_1 \\ qz_1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} qx_0 \\ qy_0 \\ qz_0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} qx_1 \\ qy_1 \\ qz_1 \\ 1 \end{bmatrix} \quad \textit{traslazione pura}$$

$$\begin{bmatrix} qx_0 \\ qy_0 \\ qz_0 \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} qx_1 \\ qy_1 \\ qz_1 \\ 1 \end{bmatrix} \quad \textit{rotazione pura}$$

La matrice omogenea non gode della proprietà di ortogonalità!

8.7 Composizione di roto-traslazioni

E' una sequenza di trasformazioni che seguono le regole sotto elencate:

- Se il sistema di riferimento $O-U-V-W$ ruota attorno ad un asse principale del sistema di riferimento $O-X-Y-Z$, allora bisogna usare la *pre-moltiplicazione*
- Se il sistema di riferimento $O-X-Y-Z$ ruota attorno ad un asse principale del sistema di riferimento $O-U-V-W$, allora usare la *post-moltiplicazione*

Esempio: rotazione di α attorno OX; traslazione di b lungo OX; traslazione di d lungo OZ; rotazione di θ attorno OZ

$$\begin{aligned}
H &= \mathbf{R}_{z,\theta} \cdot \mathbf{T}_{z,d} \cdot \mathbf{T}_{x,b} \cdot \mathbf{R}_{x,\alpha} \\
&= \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

9 Mapping 2D

Obiettivo costruire una mappa 2D dell'ambiente.

Un robot può riceve in input diverse mappe per poi utilizzarle per eseguire vari task (e.g localizzazione). Esistono diverse tipologie di mappe:

- mappe geometriche:
  - sono mappe costruite da un architetto, quindi sono molto precise e rappresentano l'ambiente in modo esatto.
  - Richiede dei sensori super belli per essere costruita
- mappe di occupazione
  - consite in una griglia, in cui le celle possono essere occupate o non occupate (probabilità).
  - ☺ buona per rappresentare aree piccole
  - ☹ pesante per aree vaste
- mappa topologica
  - si basa su grafi per descrivere l'ambiente: i nodi sono le stanze, gli archi i percorsi.
  - ☺ ottima per descrivere l'ambiente
  - ☹ può essere complessa (algoritmi sui grafi)

9.1 Problemi

Durante la fase di mappatura con i robot, ci sono diversi problemi che bisogna affrontare.

9.1.1 Rumore di misurazione

La mappatura dell'ambiente è abbastanza facile da risolvere, se il rumore nelle differenti misure fosse indipendente (basterebbe prendere tante misure per migliorare la mappa).

Purtroppo la mappatura ha degli ***errori dipendenti statisticamente***. Questo perchè il rumore è additivo e un piccolo errore, a ogni misura diventa grande nella misura finale.

Figura 52: Errori odometria.

- **Esempio 1**: se si usa l'odometria per mappare un l'ambiente dove trova il robot si avranno grossi errori di rotazione (figura 52).

- **Esempio 2**: il motion blur è un rumore che è dovuto alla visione di oggetti distorti dalla prospettiva se si sta mappando sfruttando una telecamera sugli oggetti lontani, e il tempo di acquisizione non è molto accurato.

9.1.2 Problema della corrispondenza

Il problema consiste nel capire se le misurazioni del sensore, prese in istanti temporali diversi, corrispondono allo stesso oggetto (ostacolo) nell'ambiente.

Il problema è complesso perchè alla fine del ciclo macchina, il robot ha accumulato un errore di stima sulla sua posizione attuale, che può essere molto ampio.

Pertanto è difficile capire quale ipotesi, sulla posizione attuale, è vera dato che il loro numero cresce esponenzialmente a causa dell'errore.

9.1.3 Ambiente dinamico

L'ambiente che circonda il robot è, di solito, dinamico. Quindi variando può introdurre errore ulteriore alle misure fatte con i sensori, causando così una crescita ancora più forte del numero di ipotesi attive.

9.2 Soluzioni - Probabilistic robotic mapping

Quasi tutti gli algoritmi per la mappatura utilizzano modelli probabilistici del robot e del suo ambiente, e tutti si basano sull'inferenza probabilistica per trasformare le misure dei sensori in mappe.

Il motivo della popolarità delle tecniche probabilistiche deriva dal fatto che la mappatura del robot è caratterizzata da incertezza e da rumore del sensore.

Gli algoritmi probabilistici approcciano il problema esplicitamente modellando differentemente le origini del rumore e i loro effetti sulle misure.
Vantaggi:

- è in grado di accogliere modelli non accurati e sensori imperfetti
- è robusto nelle applicazioni reali. Miglior approccio conosciuto sui peggiori problemi dei robot.

Svantaggi

- computazionalmente esigente;
- parte da assunzioni sbagliate;
- è approssimato.

9.2.1 Approcci odierni

Gli approcci odierni usano mappe probabilistiche, perchè permettono di rappresentare gli errori dovuti all'incertezza dei dati in input.

Esistono due metodologie:

- Short term mapping:
  - Si crea una mappa che viene usata solo per poco ed è ottima per fronteggiare un ambiente mutevole, si usa per schivare gli ostacoli.
- Long-Term Mapping:
  - Si crea una mappa che verrà usata per parecchio e quindi può essere usata per pianificare dei movimenti, ma richiede una risoluzione più elevata, una maggiore precisione e una mappa più grande.

9.3 Mappa geometrica 2D

Sono mappe in cui gli oggetti sono rappresentati tramite oggetti geometrici (linee e poligoni). Si possono costruire usando un telemetro laser che restituisce un insieme di punti nel piano con coordinate rappresentate tramite la posizione del robot in cui si rappresenta anche la varianza dei punti.

Trovare gli elementi

Tramite la posizione dei punti e la loro varianza si può usare un algoritmo di regressione lineare per trovare la retta che meglio li approssima. In questo modo si può rappresenta il bordo di un ostacolo, mediante la retta trovata.

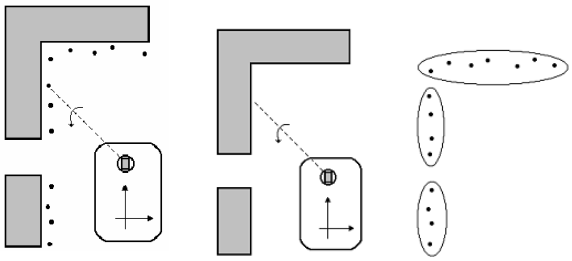


Figura 53: Riconoscimento degli ostacoli con un telemetro laser.

Ma gli ostacoli non sono sempre rettilinei e quindi bisogna raggruppare i punti i zone che sono rettilinee

$$\begin{aligned} f(x,p) &= r - x \cdot cos(\theta) - y \cdot sin(\theta) = 0 \quad p = (r,\theta) \\ f(x,p) &= a \cdot x + a \cdot y + c = 0 \qquad p = (a,b,c) \\ f(x,p) &= y - m \cdot x - q = 0 \qquad p = (m,q) \end{aligned}$$

Per raggrupparli si prendono i punti la cui distanza (in una direzione) rispetto a quello successivo è abbastanza bassa, e si crea un nuovo gruppo se la soglia è superata.

Ovviamente se questa soglia è mal posta il raggruppamento farà schifo e non si vedranno dei buchi. Parametrizzando la retta nel piano con i parametri da stimare ρ e θ

$$r - (x \cdot \cos(\theta) + y \cdot \sin(\theta)) = 0$$

si ottiene il problema di regressione lineare, allora l'errore della relazione è definito come

$$e = x \cdot \cos(\theta) + y \cdot \sin(\theta) - \rho$$

Quindi il problema della regressione lineare è impostato per andare alla ricerca dei valori dei parametri che minimizzano l'errore per l'insieme di misure considerato.

Unire le retta

Una volta trovate le varie rette bisogna unirle per comporre la mappa. Ma se si ha un ostacolo visto da diverse prospettive, bisogna capire che è lo stesso visto da diverse angolazioni e non sono due parti separate.

Questo è un problema perchè se il robot sbaglia leggermente la sua posizione le due rette non saranno perfettamente sovrapposte.

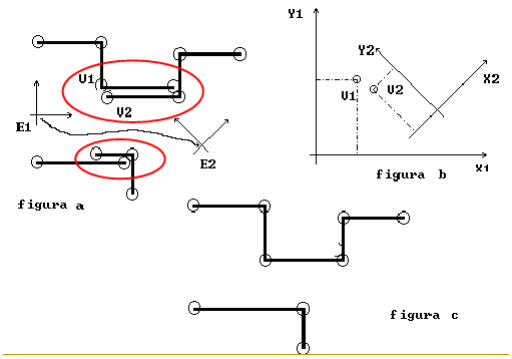


Figura 54: Composizione delle mappe locali.

9.4 Mappa di occupazione 2D

Sono mappe a griglia in cui ogni punto rappresenta la possibilità che sia presente un ostacolo. Queste mappe si possono creare usando un sensore sonar.

Forward sensor model

Effettuando una misura il sonar restituisce una funzione di probabilità

$$f(z|\rho, \theta)$$

ossia la probabilità che ci sia un ostacolo alla distanza z avendo un sonar che vede tramite un cono con altezza ρ e angolo di apertura θ.

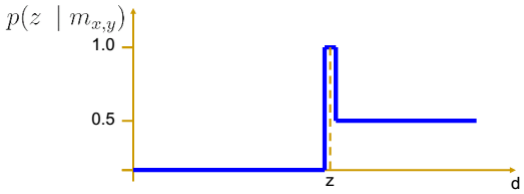


Figura 55: Probabilità di misura del sensore sonar.

Attraverso una serie di esperimenti (metodo empirico), si può trovare la bontà del sensore nell'individuare gli ostacoli e si possono definire delle distribuzioni di probabilità su alcune aree del cono.

Il cono allora viene mappato su una griglia e si assegna a ogni cella una probabilità che ci sia l'ostacolo, grazie alla misura del sonar che fornisce la probabilità per aggiornare la cella, mediante la regola di **Bayes**.

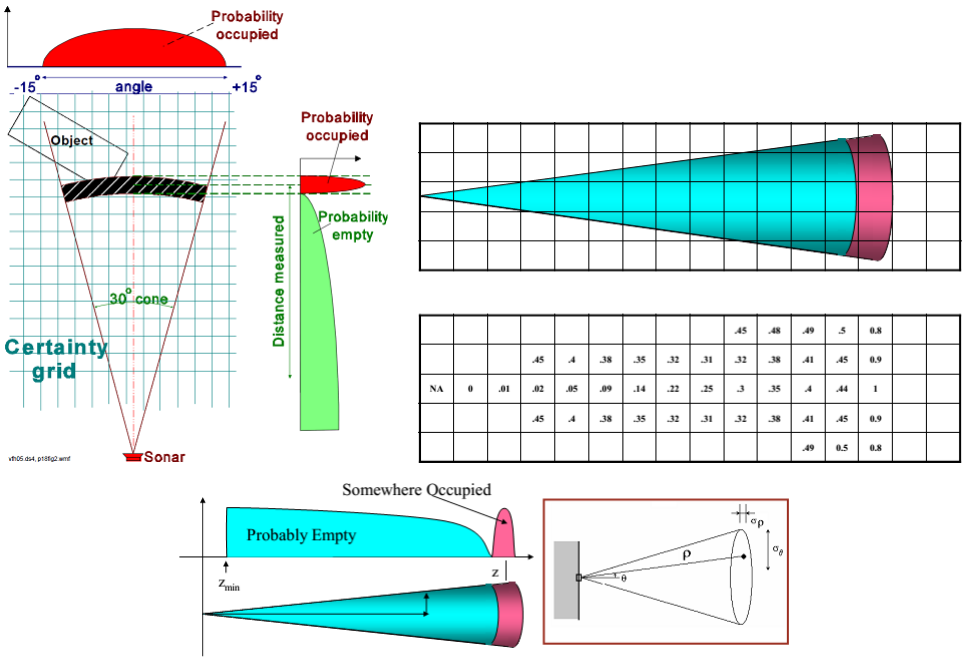


Figura 56: Modello del sensore sonar ricavato su un esperimento di k misure.

$$p(z|\rho, \theta) = \frac{1}{2\pi\sigma_\rho\sigma_\theta} \cdot e^{-\frac{1}{2}\left(\frac{(z-\rho)^2}{\sigma_\rho^2} - \frac{\theta^2}{\sigma_\theta^2}\right)}$$

Inverse sensor model

Applicando la regola di **Bayes** su una singola cella $s(C_{x,y})$, data una lettura z dal sensore si ha

$$
\begin{aligned}
p\left(s(C_{x,y}) = OCC \mid z\right) &= \frac{likelihood \cdot prior}{normlization} \\
&= \frac{p\left(z \mid s(C_{x,y}) = OCC\right) \cdot p\left(s(C_{x,y}) = OCC\right)}{\sum p\left(z \mid s(C_{x,y})\right) \cdot p\left(s(C_{x,y}) = OCC\right)}
\end{aligned}
$$

Creando una nuova mappa con

$$\sum p\left(z \mid s(C_{x,y})\right) \cdot p\left(s(C_{x,y}) = OCC\right) = 0.5$$

e

$$p\left(s(C_{x,y}) = OCC\right) = 0.5$$

si ottiene

$$p\left(s(C_{x,y}) = OCC \mid z\right) = p\left(z \mid s(C_{x,y}) = OCC\right)$$

Inizializzazione della mappa e aggiornamento della mappa

Prima di partire con la mappatura dell'ambiente, bisogna **inizializzare** le **celle della mappa** a qualche **valore** pari a **0.5**.

Durante il movimento del robot, le probabilità contenute nelle celle vengono aggiornata (mediante Bayes), grazie alle nuove misure che si effettuano con i sonar.

Il contenuto delle celle viene rappresentato con una scala di grigio, avendo

- celle libere: colore grigio chiaro
- celle occupate (ovvero quelle con probabilità che superano la soglia minima tale per qui la cella è considerata occupata): colore tendente al nero.

Figura 57: Occupancy map.

9.5 Mappa topologica 2D

Sono mappe in cui si evidenza le connessione tra varie zone tramite un grafo $G = (N, A)$ di possibili percorsi.

Ossia la mappa è un grafo in cui:

- I nodi sono posti riconoscibili e ben definiti, ossia hanno qualcosa che li distinguono dai loro vicini, non è necessario che siano distinguibili localmente.
- gli archi i percorsi che li collegano

Creazione della mappa

Per creare la mappa il robot deve:

- Tramite i sensori capire se si trova in un posto distinto, che diventerà un nodo, ma farlo non è semplice perchè si possiedono solo info locali. Per farlo il sensore assegna a ogni posto delle caratteristiche che può osservare tramite dei sensori.
- Si muove in una direzione possibile e controlla se un posto ha le caratteristiche necessario per essere un nodo.
- Trovato il nodo controlla se quel nodo è già sulla mappa e lo collega con il percorso fatto al nodo in cui si trovava prima altrimenti ne crea uno nuovo e fa la stessa cosa.

10 Robot Localization

Obiettivo Il robot deve capire dove si trova, ma dato che l'odometria è imprecisa si usano altri sensori per migliorare la stima e lasciare all'odometria la stima solo su piccoli tragitti.

10.1 Probabilistic Map-Based Localization

Idea

Data una mappa precisa dell'ambiente, il robot con i sui sensori è in grado di *leggere* la mappa intorno a lui.

L'idea è quella di trovare la rototraslazione necessaria per sovrapporre la mappa locale appena creata con i sensori, con quella globale. Se si trova un punto che coincide perfettamente allora quella rototraslazione si può anche applicare al robot per capire la sua posizione rispetto alla mappa.

Problemi

Fare il procedimento di matching delle due mappe, avendo una sola misura della mappa è impossibile. Questo perchè la mappa creata è sicuramente imprecisa (*sensor aliasing*: informazione insufficiente dai sensori, con un solo colpo di scannerizzazione, e soggetta a errori) e inoltre potrebbero esserci più posti simili nella mappa.

Pertanto è necessario fare più misure della mappa, attraverso i sensori, e continuare ad aggiornare la stima della postizione del robot in modo probabilistico.

Concetto per la realizzazione

Il robot esegue ciclicamente due azioni:

- Action update

  Che aggiorna la stima della posizione al passo precedente s_{k-1} tramite l'odometria o_k, ottenendo la nuova sitma $s_k^{'}$ per la posizione attuale. Questo aggiornamento aumenta l'errore della stima perchè l'odometria non funziona bene.

$$s_k^{'} = Act\left(o_t, s_{k-1}\right)$$

- Perception update

  Che aggiorna la stima $s_k^{'}$ calcolata usando l'odometria in base alla misura della mappa i_k effettuata con il sensore laser, ottenendo così la stima s_k. Questo aggiornamento riduce l'errore introdotto al passo prima (action update).

$$s_k = See\left(i_k, s_k^{'}\right)$$

Quindi prima si predice la posizione usando i dati dell'odometria, e successivamente li si paragona a quelli misurati dai sensori in modo da creare una stima migliore ottenuta dalla fusione delle due informazioni.

10.1.1 Passi della map-based localization

Questa tipologia di localizzazione segue una cinque passi come mostrato in figura 58. Siano dati

- $p\left(k|k\right)$: la stima della posizione
- $\sum p\left(k|k\right)$: la covarianza della stima per l'istante k
- $u\left(k\right)$: l'azione di controllo nell'sitante attuale
- $Z\left(k+1\right)$: l'insieme delle osservazioni
- $M\left(k\right)$: la mappa dell'ambiente in cui è il robot

Si calcola

- $p\left(k+1\,|\,k\right)$: la stima della nuova posizione (a posteriori)
- $\sum p\left(k+1\,|\,k\right)$: la covarianza della nuova stima

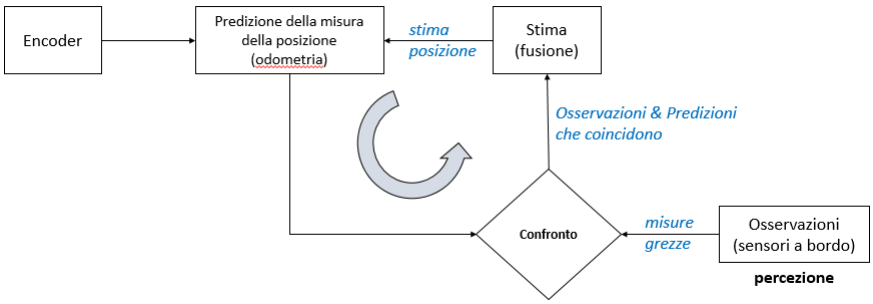


Figura 58: Passi della localizzazione basata su mappe.

10.1.2 Monte Carlo Localization using Particle Filtering

Il metodo Monte Carlo considera un numero N di ipotesi (*particles*) sulla posizione S del robot. *L'utlizzo di più ipotesi contemporaneamente evita il problema dei max/min locali.*

A ogni ipotesi s_i assegniamo una probabilità (peso) p_i corrispondente alla posizione $l_i = (x, y, \theta)$ del robot (all'inizio dell'algoritmo le ipotesi sono equiprobabili).

$$\begin{aligned} S &= \{s_i \,|\, i = 1 \dots N\} \\ s_i &= [l_i, p_i] \end{aligned}$$

A ogni iterazione del filtro, queste particelle si possono spostare e variare la loro probabilità in modo da avere sempre una indicazione della posizione del robot più o meno accurata.

- Action update

  Si parte dalla stima al passo precedente e la si aggiorna con l'odometria.

  Dato che l'odometria ha una grossa varianza, il filtro sparge le particelle di più rispetto a prima indicando l'*incertezza della stima*.

$$p(l \,|\, l, a)$$

- Perception update

  Una volta ottenute le nuove particelle si può confrontare quello che il robot vedrebbe se si trovasse in quella determinata particella e paragonarlo a quello che effettivamente vede.

  Così si può cambiare la probabilità di quella particella e quindi migliorare la stima.

  Sia data la particella

$$s_i = [l_i, p_i]$$

  allora

$$p_i = \alpha \cdot p\,(m \,|\, l)$$

  dove m è una misurazione del sensore e α una costate di normalizzazione così la somma delle $p = 1$.

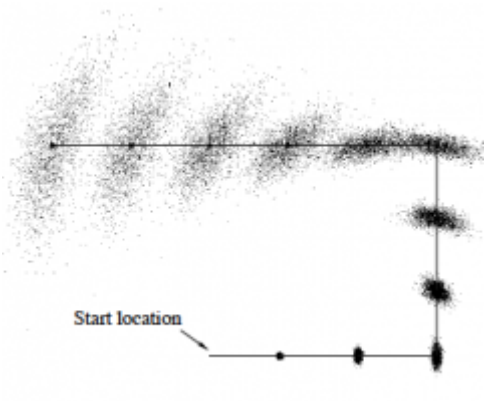


Figura 59: Ipotesi di Monte Carlo Localization.

Durante l'esecuzione dell'algoritmo, si utilizzano due modelli diversi:

- modello del moto, durante la fase di predizione;
- modello del sensore, durante la fase delle osservazioni (misurazioni).

Il numero delle ipotesi fatte deve essere un numero significativo, che varia però drasticamente in base alla situazione in cui si trova il robot.

Durante la fase di **localizzazione globale** il robot non conosce nulla dell'ambiente quindi ha bisogno ti **tante ipotesi**, che verranno poi diminuite durante l'esplorazione.

Durante la fase di **position tracking** l'incertezza del robot è piccola dato che si hanno delle stime antecedenti (meno varianza sulla posizione del robot), quindi il numero di ipotesi attive è piccolo.
MCL decide il numero di ipotesi attive in run-time:

- confronta la probabilità $p(l)$ della posizione prima della misurazione, con quella $p(l \mid s)$dopo la misurazione
- decide il numero di ipotesi attive
    - più c'è varianza più particelle tiene

Più ipotesi attive si hanno, più oneroso è l'algoritmo in temrini di memoria e di calcolo.

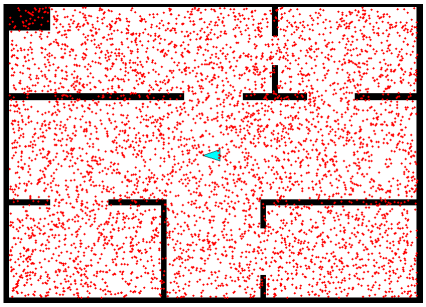


Figura 60: Ipotesi di Monte Carlo Localization.

10.2 Marker-based Localization and Navigation

Il robot ha a disposizione una mappa di marker univocamente identificabili dai suoi sensori, per capire la propia posizione.

I marker possono essere anche oggetti dell'ambiente noti e non solo etichette personalizzate.

10.2.1 Triangolazione

Individuando due marker si forma un triangolo tra la posizione del robot e i due marker. Allora tramite tecniche di triangolazione si può stimare la posizione del robot.

Per farlo sono necessari dei sensori in grado di fare una di queste 3 cose:

- Individuare la distanza tra i due marker e la distanza tra il robot e i due marker, ossia avendo tutti i 3 lati del triangolo.
- Individuare la distanza tra i due marker e l'angolo tra di loro, ossia avendo 2 lati e 1 angolo.
- Individuare la distanza di un marker e l'angolo tra quello e il robot e l'altro marker.

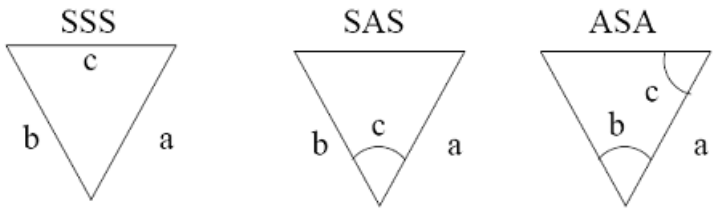


Figura 61: Ipotesi di Monte Carlo Localization.

Marker circolari

Il robot conosce la posizione dei marker $L_i\,(x_i, y_i)$. Questi possono trasmettere al robot la distanza r_i tra loro e il robot stesso, ma non la direzione in cui è guardato.

Pertanto il robot sa solo di essere dentro una circonferenza.

Se riceve l'informazione da due marker può incrociare le due circonferenze per trovare il punto in cui si trova, ma dato che la distanza è afflitta da errore possono esserci più punti di sovrapposizione tra le due circonferenze

$$
\begin{aligned}
r_1 &= \sqrt{(x - x_1)^2 + (y - y_1)^2} \\
r_2 &= \sqrt{(x - x_2)^2 + (y - y_2)^2}
\end{aligned}
$$

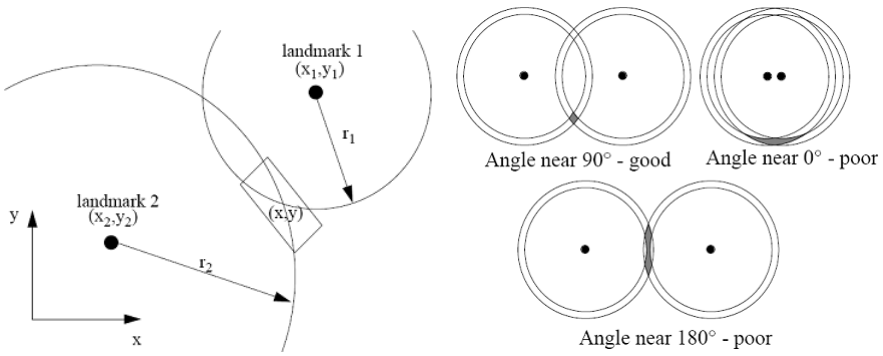


Figura 62: Ipotesi di Monte Carlo Localization.

Segnaletica orizzontale

Si mettono marker a terra per rappresentare un percorso ben definito dai simboli a terra. Questi marker possono contenere informazione (e.g. i twist che il robot deve eseguire, ecc).

10.2.2 Triangolazione: posizione angolare del robot nota

Suppoendo che:

- X e Y indicano la posizione del robot rispetto al mondo, parametro incognito.

- θ_1 è la posizione angolare del primo marker rispetto al mondo, parametro incognito (dipende dalla posizione del robot).
- θ_2 è la posizione angolare del secondo marker rispetto al mondo, parametro incognito (dipende dalla posizione del robot).
- x_1 e y_1 sono le posizioni rispetto al mondo del primo marker.
- x_2 e y_2 sono le posizioni rispetto al mondo del primo marker.

è possibile misurare θ_1 e θ_2 se si possiede un sensore che stima la posizione rispetto al mondo del robot (una bussola), in questo caso si possono usare le seguenti formule, che sono sempre valide:

$$\begin{cases} \dfrac{\sin\theta_1}{\cos\theta_1} = \dfrac{y_1 - Y}{x_1 - X} \\ \dfrac{\sin\theta_2}{\cos\theta_2} = \dfrac{y_2 - Y}{x_2 - X} \end{cases}$$

da cui si possono ricavare dei sistemi lineari facilmente risolvibili:

$$\begin{cases} \sin\theta_1 \cdot (x_1 - X) = \cos\theta_1 \cdot (y_1 - Y) \\ \sin\theta_2 \cdot (x_2 - X) = \cos\theta_2 \cdot (y_2 - Y) \end{cases}$$

che sono sempre risolvibili a meno che la retta che unisce i due marker passa esattamente nella posizione del robot.

Questo sistema presenta una imprecisione per due motivi:

- A volte il sistema non è risolvibile.
- I sensori che misurano hanno una incertezza e quindi i marker sono localizzati non perfettamente. In generale se si trova la posizione di un marker usando un sensore che vede con un cono allora più il marker è lontano peggio è, dato che il cono si allarga.

10.2.3 Triangolazione: posizione angolare del robot sconosciuta

In questo sistema anche θ del robot è incognita e quindi sono necessari tre marker e le equazioni sono analoghe a quelle di prima:

$$\begin{cases} \dfrac{\sin(\theta + \theta_1')}{\cos(\theta + \theta_1')} = \dfrac{y_1 - Y}{x_1 - X} \\ \dfrac{\sin(\theta + \theta_2')}{\cos(\theta + \theta_2')} = \dfrac{y_2 - Y}{x_2 - X} \\ \dfrac{\sin(\theta + \theta_3')}{\cos(\theta + \theta_3')} = \dfrac{y_3 - Y}{x_3 - X} \end{cases}$$

dove θ_i' è la posizione angolare del marker rispetto al robot.

11 Mapping 3D

Obiettivo Tecnica di costruzione di mappe 3D dell'ambiente, partendo da acquisizioni di misure di distanze o attraverso diverse tipologie di sensori.

11.1 Point Cloud

La nuvola di punti, è un insieme di vertici posti in un sistema di riferimento tridimensionale. Ogni vertice è espresso con tre coordinate (x, y, z). Rappresentano un sottoinsieme dei punti appartenenti alla superficie di un oggetto (a volte si usano anche per memorizzare informazioni sul colore).

11.2 Sistemi di misura

11.2.1 Scanner 3D

Sono dispositivi che acquisiscono informazioni relative alla distanza degli oggetti presenti nel loro campo visivo. Per ogni punto rilevato fornisce informazioni in coordinate sferiche, che possono essere convertite anche in coordinate cartesiane.

$$\begin{cases} x = r \cdot \sin(\theta) \cdot \cos(\phi) \\ y = r \cdot \sin(\theta) \cdot \sin(\phi) \\ z = r \cdot \cos(\theta) \end{cases} \qquad \begin{cases} r = \sqrt{x^2 + y^2 + z^2} \\ \theta = \arccos\left(\frac{z}{r}\right) \\ \phi = \arctan\left(\frac{y}{x}\right) \end{cases}$$

11.2.2 Time of flight scanner

È uno scanner di tipo attivo, che emette un raggio luminoso (ultrasuoni o x-ray) e calcola la distanza in base al tempo che il raggio riflesso impiega per tornare all'origine.

Nota la velocità del raggio c, la distanza vale

$$d = \frac{c * t}{2}$$

Per ogni misura viene emesso un solo raggio, quindi la misura dell'intero campo visivo e sequenziale, e inoltre si possono misurare dai 10.000 ai 100.000 punti al secondo (usato per aqcuisire informazioni su grandi edifici).

11.2.3 Scanner basati su triangolazione

È uno scanner di tipo attivo che utilizza un laser ed una telecamera.

Il laser emette un raggio che illumina l'oggetto, mentre la telecamera viene utilizzata per riprendere il punto in cui il laser colpisce l'oggetto.

Sono noti la distanza tra il laser e la telecamera e l'angolo di inclinazione del laser. L'angolo della camera può essere calcolato sfruttando la posizione del punto ripreso nel campo visivo.

Pertanto triangolando è possibile calcolare tutte le misure del triangolo e quindi la posizione del punto illuminato. Affinche si velocizzino le operazioni si può emettere una linea luminosa anziché un solo punto.

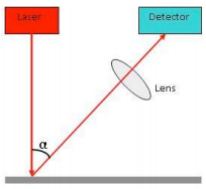


Figura 63: Principio di triangolazione del laser con la telecamera.

11.2.4 Structured light scanner (Microsoft Kinect)

È uno scanner di tipo attivo. Consiste in un proiettore luminoso, che emette un fascio di luce con un particolare pattern, e in una telecamera che riprende l'oggetto.

- Pattern 1D: ***linea***
  - La linea proiettata si deforma in base alla superficie dell'oggetto.
  - Attraverso la linea distorta ripresa dalla camera è possibile triangolare e calcolare la posizione dei punti.
- Pattern 2D: ***griglia***
  - Sono più veloci degli scanner basati su triangolazione di un punto e ossono acquisire informazioni su oggetti in moto.

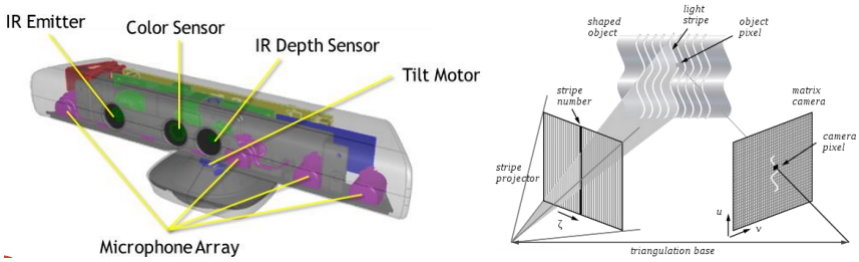


Figura 64: Kinnect e il suo principio di triangolazione del laser.

In particolare kinect è in grado di leggere una immagine RGB normale e assegnare a ogni pixel la profondità.

Inoltre il dispositivo riesce a riconoscere anche oggetti in movimento.

11.3 Visione stereoscopica

La visione stereoscopica punta a ricostruire la profondità a partire da due immagini scattate allo stesso oggetto da due telecamere poste in posiziono doverse, nota a priori. Per farlo si deve riconoscere l'oggetto su entrambe le immagini e poi fare la triangolazione per stabilirne la distanza.

Il problema può essere scomposto in due sottoproblemi: *calcolo delle corrispondenze* e *triangolazione*.

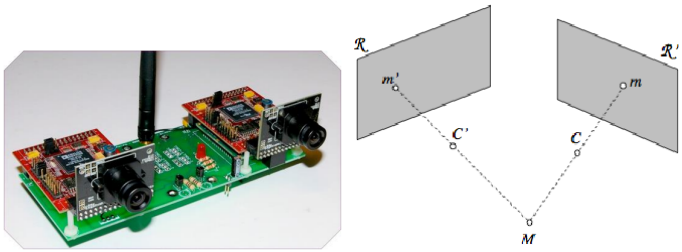


Figura 65: Visione stereoscopica e triangolazione

11.3.1 Triangolazione

Siano date le seguenti definizioni:

- **Baseline**: Asse che unisce le due telecamere.
- **Piano epipolare**: Piano che passa per la baseline.
- **Geometria epipolare**: Geometria che studia i piani epipolari.
- **Epipolo**: Punto di intersezione della baseline con il piano immagine.

- **Linea epipolare**: L'intersezione tra un piano epipolare e il piano immagine (che è una retta) l'intersezione tra la linea epipolare e la baseline è sempre l'epipolo.

- **Linea corrispondente**: Per ogni punto x di una delle immagini si definisce la sua linea corrispondente dell'altra immagine e si sa che il punto in cui è ripreso x nell'altra immagine si trovi su tale retta.

Tramite la geometria epipolare è possibile vincolare la ricerca del punto in comune a dei punti appartenenti a una certa retta e non a tutta l'immagine.

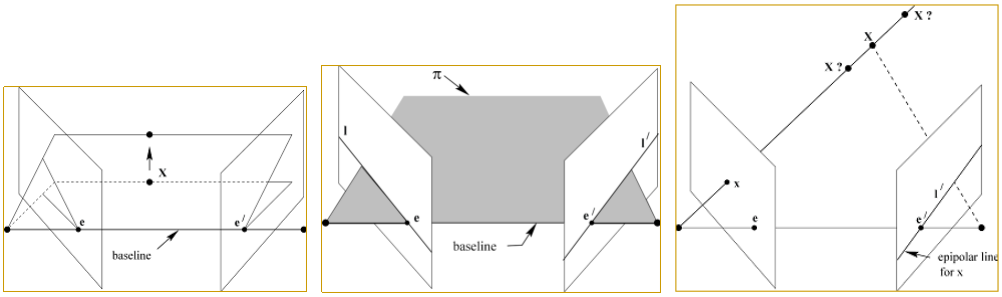


Figura 66: Visione stereoscopica e triangolazione

11.4 Compressione dei dati

Obbiettivo a volte i sensori danno più informazione di quella che serve (superifici irregolari degli oggetti) e quindi è necessario comprimerla. In questo modo si riduce l'informazione su cui lavorano gli algoritmi di elaborazione delle immagini.

11.4.1 Algoritmo Octree

L'informazione è archiviata detro un albero in cui ogni nodo (rappresenta un pezzetto di spazio) dà 8 figli.

Si procede ricorsivamente nel seguente modo:

- Se lo spazio rappresentato da quel nodo è abbastanza uniforme secondo alcuni criteri definibili (colore o profondità) si assegna questo valore al nodo e si termina la ricorsione.

- Altrimenti si divide lo spazio in 8 parti uguali e si assegnano agli 8 nodi figli.

Si procede così finchè tutte le celle non rispettano il criterio di similitudine o si raggiunge una profondità massima.

11.4.2 Segmentazione

Il processo di segmentazione divide i punti di una point cloud in diversi sottoinsiemi.

Ognuno di questi sottoinsiemi contiene i punti di un particolare oggetto.
Per determinare le varie regioni:

- si calcola il vettore normale a ogni punto sfruttando i punti nel vicinato e i sottoinsiemi sono determinati dalla distanza e dalla comparazione dei vettori normali dei vari punti.

- usando modelli di riferimento (non spiegati)

- decomposizione in sottocelle e merging (non spiegati)

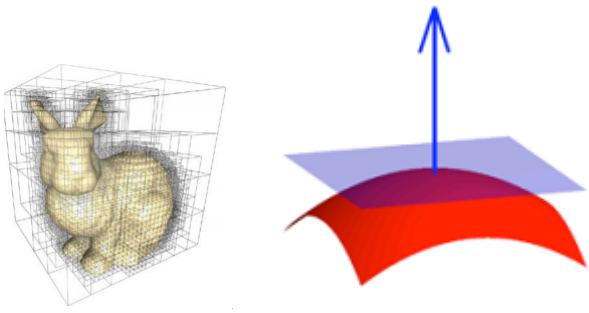


Figura 67: Suddivisione dell'immagine in cubi e segmentazione usando una normale.

11.4.3 Generazione di meshes

Il processo di generazione di meshes punta a trasformare una nuvola di punti in un insieme di poligoni, generalmente triangoli. Questi poligoni rappresentano l'approssimazione della superficie dell'oggetto da ricostruire.

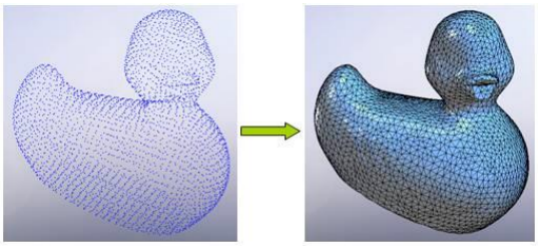


Figura 68: Generazione di meshes.

12 Simultaneous Localization and Mapping (SLAM)

Obbiettivo Si vuole realizzare un metodo che permetta a un robot autonomo di generare una mappa autonomamente e usarla in run-time per la localizzazione.

Il problema non è altro che la composizione di due sottoproblemi: *localizzazione* e *mappatura*, che derivano entrambi dalle misurazioni dei sensori.

La soluzione è data considerando inisieme i sottoproblemi.

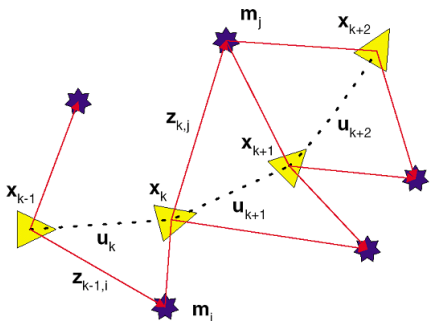


Figura 69: Elementi di SLAM.

Elementi:

- $k = 1, 2, ..., n$ indice di tempo discreto.
- $\mathbf{x}_k$ vettore delle posizioni vere del robot, all'istante k.
- $\mathbf{u}_k$: vettore di controllo dato al robot (twist), all'istante $k-1$ per portarlo da x_{k-1} a x_k all'istante k.
- ★ marcatore o qualsiasi oggetto facilmente distinguibile.
- m_i la posizione vera del marker i-esimo.
- $z_{k,j}$ misura della distanza dal j-esimo marcatore, effettuata della posizione x_k all'istante k.
- z_k una generica misura (di uno dei marcatori) effettuata all'istante k.

Parametri:

- Storico degli stati: $X^k = x_0, x_1, \ldots, x_k = X^{k-1}, x_k$
- Storico degli input di controllo: $U^k = u_1, u_2, \ldots, u_k = U^{k-1}, u_k$
- Insieme dei marcatori: $m = m_2, m_2, \ldots, m_M$
- Storico delle osservazioni: $Z^k = z_1, z_2, \ldots, z_k = Z^{k-1}, z_k$

Se il robot ha a disposizione una mappa $\rightarrow$ *riesce a localizzarsi*

Se il robot riesce a localizzarsi $\rightarrow$ *può costruire la mappa dell'ambiente che lo circonda*

12.1 Localizzazione

Dalle osservazioni Z^k, si possono fare inferenze sulla posizione X^k del robot e/o inferenze sulle posizioni m dei marcatori.

Una mappa (geometrica, di marcatori, di occupazione) m è nota a priori, e da U^k si fanno inferenze sulle posizioni *sconosciute* X^k.

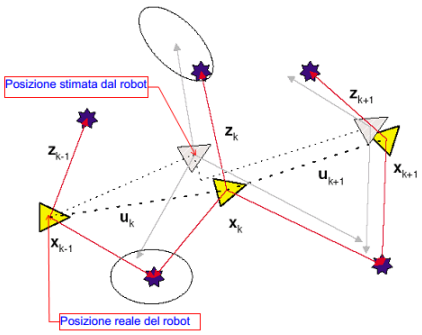


Figura 70: Problema della localizzazione.

12.2 Mapping

In questo problema le posizioni X^k del robot sono note e si vuole calcolare la posizione dei vari marker, per costruire una mappa m.

A ogni istante si aggiorna la stima della posizione del robot sfruttando le misure delle varie distanze, dato che si conosce esattamente la posizione del robot e la varianzia delle misure.

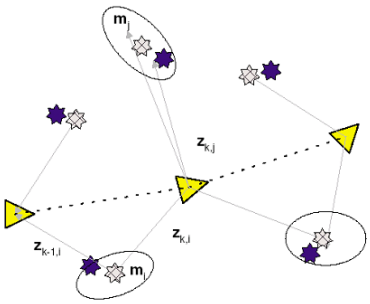


Figura 71: Problema della mappatura.

12.3 SLAM

Il robot non ha nessuna informazione circa l'ambiente che lo circonda. Si assume nota la posizione iniziale x_0 del robot e la sequenza di comandi U^k dati in input al robot.

Nello SLAM si applicano le tecniche di mappatura e di localizzazione contemporaneamente, cercando di milgiorare entrambe le sitme ad ogni passo mediante inferenze.

Grazie ai sensori montati a bordo il robot è in grado di corstruire una mappa m (il tipo di mappa può dipendere dal tipo di sensore), ovviamente la bontà della stima dipende dalla precisione del sensore.

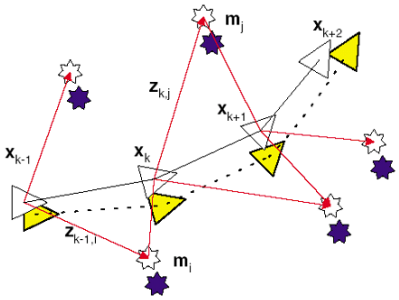


Figura 72: Problema dello SLAM.

Osservando dal robot lo stesso marcatore di cui si conosce esattamente la posizione è possibile migliorare la stima della posizione del marcatore dato che si conosce l'incertezza.

12.4 Metodo Bayesiano approssimato per lo SLAM

Questo metodo sfrutta la stima bayesiana per risolvere lo SLAM.

Si usa Bayes perchè si vogliono stimare due informazioni, tramite una sola misura, quindi le due stime vanno fatte insieme.

Per fare ciò bisogna dare una modellizzazione probabilistica del sistema.

- il modello del ***sensore*** viene visto come una verosimiglianza $p(z_k|x_k, m)$
  - probabilità che il sensore misuri la distanza z_k dal marcatore m dalla posizone x_k (stimabile sperimentalmente)
  - può essere vista al contrario come la probabilità di essere alla posizione x_k se di misurare z_k.
- il modello del ***movimento*** del robot è considerato come una probabilità condizionata $p(x_k|x_{k-1}, u_k)$
  - probabilità che il robot raggiunga la posizione x_k partendo da x_{k-1} con un twist u_k.

Si stima poi ricorsivamente anche

$$p(x_k, m|Z^k, U^k, x_0)$$

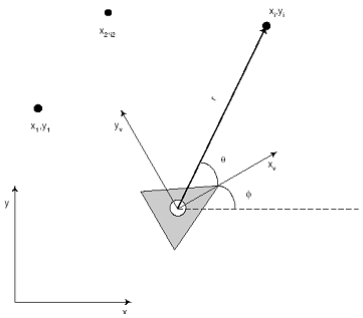
Si assume inoltre anche il modello del robot sia markoviano

$$p(x_k|x_{k-1}, u_k) = p(x_k|x_{k-1}, u_k, X^{k-2}, U^{k-1}, m)$$

.

12.4.1 Extended Kalman Filter (EKF)

E' uno stimatore lineare ricorsivo per sistemi descritti da processi non lineari, ed è alla base di quasi tutti gli algoritmi SLAM.



$$\begin{aligned} Vehicle\ state\ \mathbf{x}(k) &= [x(k), y(k), \phi(k)]^T \\ Vehicle\ control\ \mathbf{u}(k) &= [V(k), \psi(k)]^T \end{aligned}$$

Figura 73: Modelli di moto e di osservazione.

EKF descrive il **modello di moto del veicolo** in termini di modello cinematico affetto da errore gaussiano a media nulla, nella forma

$$P(x_k|x_{k-1}, u_k) \Leftrightarrow x_k = f(x_{k-1}, u_k) + w_k$$

$f(\circ)$ modellizza la cinematica del robot e w_k i disturbi del moto. Modello del moto, con T il tempo

$$\mathbf{x}(k) = f(\mathbf{x}(k-1), \mathbf{u}(k), k) + \mathbf{v}(k)$$

$$\begin{bmatrix} x(k) \\ y(k) \\ \phi(k) \end{bmatrix} = \begin{bmatrix} x(k-1) + T \cdot V(k) \cdot \cos(\phi(k-1) + \psi(k)) \\ y(k-1) + T \cdot V(k) \cdot \sin(\phi(k-1) + \psi(k)) \\ \phi(k-1) + T \cdot \frac{V(k)}{B} \sin(\psi(k)) \end{bmatrix} + \begin{bmatrix} q_x(k) \\ q_y(k) \\ q_\phi(k) \end{bmatrix}$$

EKF descrive anche il **modello di osservazione** in termini di osservazione geometrica affetto da errore gaussiano a media nulla, nella forma

$$P(z_k|x_k, m) \Leftrightarrow z(k) = h(x_k, m) + v_k$$

Dove $h(\circ)$ descrive la geometria delle osservazioni e $v_k$gli errori delle osservazioni. Modello della misurazione

$$\begin{aligned} \mathbf{z}(k) &= h\left(x(k)\right) + \mathbf{w}(k) \\ \begin{bmatrix} z_r^i(k) \\ z_\theta^i(k) \end{bmatrix} &= \begin{bmatrix} \sqrt{\left(X_i - x(k)\right)^2 + \left(Y_i - y(k)\right)^2} \\ \arctan\left(\frac{Y_i - y(k)}{X_i - x(k)}\right) - \phi(k) \end{bmatrix} + \begin{bmatrix} r_r^i(k) \\ r_\theta^i(k) \end{bmatrix} \end{aligned}$$

- $z_r^i(k)$ la misura radiale del sensore;
- $z_\theta^i(k)$ la misura dell'angolo;
- $(x(k), y(k), \phi(k))$ posizione del robot;
- (X_i, Y_i) posizione del marcatore *i-esimo.*

Si hanno quindi 5 incognite e 2 informazioni ($z_r^i(k)$ e $z_\theta^i(k)$), allora si effettuano n misure e per stimare si usa il EKF.

$$P\left(x_k, m|Z^k, U^k, x_0\right) = \underbrace{K}_{guadagno} \cdot \underbrace{P(z_k|x_k, m)}_{Innovazione} \cdot \underbrace{\int P(x_k|x_{k-1}, u_k) \cdot P\left(x_{k-1}, m|Z^{k-1}, U^{k-1}, x_0\right) \cdot dx_{k-1}}_{Predizione}$$

La chiave della semplicità dell'EKF è che il prodotto di distribuzioni gaussiane, dà come risultato ancora una distribuzione gaussiana.

12.5 SLAM: Stato aumentato

Lo SLAM rappresenta lo stato esteso usando sia le informazioni dello stato del robot che quelle dei marcatori.

- modello del veicolo
  - $\mathbf{x}_v(k) = [x(k), y(k), \varphi(k)]^T \quad \mathbf{u}(k) = [\omega(k), \gamma(k)]^T$
- modello dei marcatori
  - $\mathbf{m}_i = [x_i, y_i]^T$
- modello dello stato aumentato
  - $\mathbf{x}(k) \triangleq \begin{bmatrix} \mathbf{x}_v(k) \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_M \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{x}_v(k-1), \mathbf{u}(k)) \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_M \end{bmatrix} + \begin{bmatrix} \mathbf{q}_v(k) \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$
  - i marcatori sono considerati stazionari

12.6 Conclusioni EKF & SLAM

La covarianza tra le gaussiane ci dice tutto il necessario sull'errore nel processo SLAM, quindi con l'aumentare delle osservazione le informazioni della mappa aumentano.
All'aumentare delle misure il determinante della matrice delle covarianze della mappa descresce in modo montono, ma all'aumentare delle misure anche le informazioni nella mappa aumentano in modo monotono.

Inolte la correllazione tra i marcatori aumenta e quindi cresce la bontà della posizione dei marcatori locali.

$$det(\mathbf{P}(k|k)) = det(\mathbf{P}(k|k-1) - \mathbf{W}_i(k)\mathbf{S}_i(k)\mathbf{W}_i^T(k)) \leq det(\mathbf{P}(k|k-1))$$

Limite di riduzione più basso nella matrice delle covarianze della mappa.

$$\lim_{k\to\infty} det(\mathbf{P}_{mm}(k|k)) = 0$$

All'aumentare del numero di misure effettuare, nel limite gli errori nella stima della posizione dei marcatori diventano completamente correlati.

Quindi l'informazione della posizione relativa dei marcatori aumenta, e nel limite diventa esatta.
La posizione assoluta dei marcatori nella mappa è limitata dall'incertezza inziale del veicolo $\mathbf{P}_{vv}(0|0)$.

Quindi anche la stima della posizione del robot è limitata.

13 Monocular SLAM

Obiettivo Si vuole fare lo SLAM avendo solo una telecamera montata sul robot in movimento (stima la posizione degli oggetti e gli spostamenti).

Il processo consente di individuare, se possibile, punti uguali in immagini diverse per aiutare la localizzazione.

Idea

Si sfruttano i concetti della visione telescopica in cui invece delle due camere c'è una camera che si muove (movimenti piccoli).

In questo modo si ottengono delle coppie coniugate di punti che rappresentano la stessa cosa su diverse immagini.

Usando la disparità, ossia il vettore spostamento tra i due punti coniugati, si può ricostruire la mappa.

Problemi

- false corrispondenze
- occlusioni: alcuni particolari compaiono in una sola immagine;
- distorsione radiometrica: intensità dei punti corrispondenti nelle 2 immagini può essere differente;
- distorsione prospettica: lo stesso oggetto assume forme diverse nelle due immagini.

Accoppiamento feature-based

Cerca nell'immagine delle caratteristiche salienti (feature) come ad esempio: spigoli, angoli, segmenti rettilinei o curvi, daccoppiare

Questo sistema è rapido, ma fornisce mappe sparse che poi vanno unite e inoltre se la feature non è descritta bene escono brutte cose.

13.1 Algoritmo ICP (Iterative Closest Point)

L'algoritmo opera su due point cloud M (modello) e S (scene) ed esegue iterativamente tre operazioni:

1. Cerca le coppie di punti corrispondenti tra M e S.
2. Stima la trasformazione tra ogni coppia di punti.
3. Applica a tutto S le trasformazioni ottenute e valuta quale minimizza l'errore complessivo tra le varie coppie.

L'algoritmo cerca la trasformazione che minimizza l'errore complessivo di corrispondenza tra le coppie.

Svantaggi

- La sua convergenza dipende dalla qualità della stima della trasformazione.
- È assicurata la convergenza ad un minimo locale, ma non globale ⇒non è ottimo.

13.2 Optical Flow

L'optical flow si occupa di stimare il movimento di un pixel tra due immagini. La soluzione del problema di corrispondenza sta nel cercare un pixel dell'immagine H di colore c, nei dintorni dei pixel dello stesso colore c nell'imagine I.

Prima di procedere bisogna fare due considerazioni:

1. colore costante: un pixel dell'immagine H ha lo stesso colore anche nell'immagine I
2. piccoli movimenti: i punti che si cercano non si spostano di molto (u e v nella figura 74 sono meno di 1 pixel)
   (a) per garantire il piccolo spostamento una soluzione è scalare la risoluzione dell'immagine.

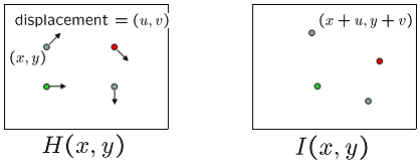


Figura 74: Movimento pixel in immagini simili

Se il movimento non è *piccolo* allora si usa una ricerca «discreta», definendo la dimensione di una finestra attorno al pixel dell'immagine H e ricercando nell'immagine I all'interno di uno spazio pari alla dimensione della finestra definita.

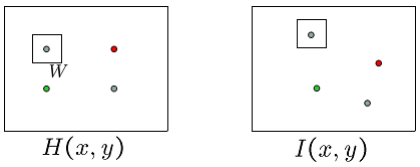


Figura 75: Movimenti *non piccoli* del pixel.

Un piccolo spostamento si può ottenere in due modi:

- acquisizione in alta frequenza
- movimenti a bassa velocità

Il lavoro può essere fatto anche ragionando su aree di pixel, invece che singoli pixel. Questo lo si può ottenere applicando dei filtri alle immagini acquisite, per ridurre la risoluzione.

Si ha quindi un insieme di pixel che corrispondono al nuovo pixel ottenuto per riduzione della risoluzione.

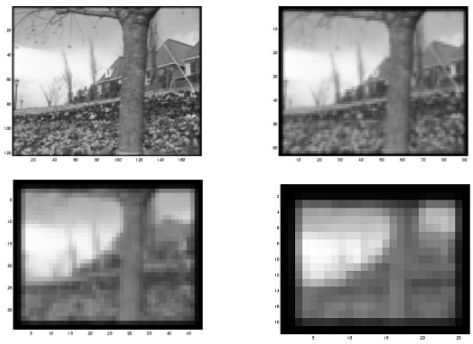


Figura 76: Movimenti *non piccoli* del pixel.

13.2.1 Stima coarse2fine

Si effettua la stima dello spostamento dei pixel su diversi livelli, partendo dal più basso (più grezzo) verso quello più alto.

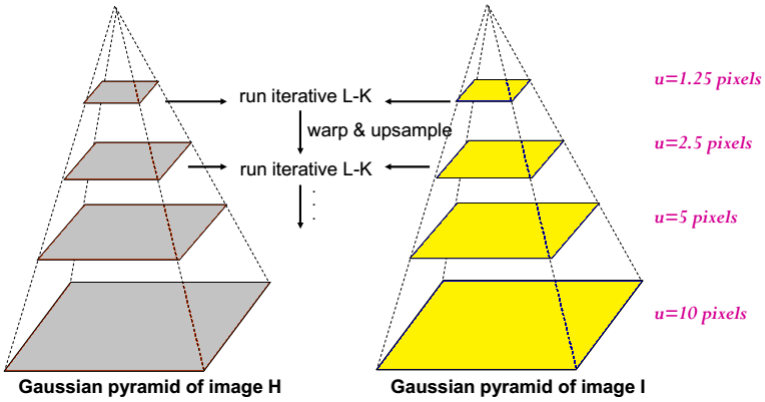


Figura 77: Stima Coarse 2 Fine

13.3 Conclusioni

- I metodi basati su feature offrono maggiore precisione
- I metodi diretti permettono di costruire una mappa semi-densa
- Alcune caratteristiche della camera influenzano in maniera considerevole i risultati raggiungibili
- Problema del calcolo della scala

14 Automated Planning

Pianificazione Significa valutare le opzioni a disposizione e tra queste scegliere la migliore, basandosi su qualche criterio (consumo energetico, distanza del percorso, autonomia, ecc).

La pianificazione, più in dettaglio può essere anche vista come un approccio flessibile per prendere decisioni complesse: lo schedule di una linea di produzione, movimenti di una gru, azioni di un robot.

Piano è una collezione di azioni da eseguire in sequenza per raggiungere un determinato obbiettivo.

Problemi

- La difficoltà è data dal poter esprimere l'obbiettivo in un linguaggio generico e trovare la soluzione automaticamente.
- I sensori sono imperfetti e si ha una conoscenza imprecisa del mondo.
- Il mondo cambia.
- il mondo ha un comportamente statistico.
- Si può sbagliare a eseguire il piano.
- la pianificazione non è immediata, bisogna perdere tempo a calcolare il piano.

14.1 State Transition System

Macchina a stati Si crea un automa a stati finiti in cui ogni stato rappresenta una possibile situazione in cui si può trovare il sistema.

In questo modo si possono stabilire gli archi come le azioni che si possono fare da una determinata situazione per raggiungerne un'altra.

In termini più rigorosi un sistema è composto dalla quadrupla $\Sigma = (S, A, E, \gamma)$ dove:

- $\Sigma = (S, A, E, \gamma)$ sistema reale;
- $S = s_0, s_1, \ldots, s_n$ stati possibili nell'ambiente;
- $A = move_1, move_2, put, take, load, unload$ azioni eseguibili dal sistema;
- $E = e_1, e_2, \ldots, e_n$ eventi esogeni, cioè azioni non sotto il controllo del pianificatore (e.g. rottura cavo della gru)
- γ funzione stato-transizione: insieme di triple$(s0, s1, a)$ con $s0, s1 \in S$ e $a \in A$

Schema a blocchi Un sistema che utilizza questo metodo è composto da 3 blocchi:

- Il sistema stesso che riceve azioni da eseguire e restituisce delle osservazioni date dai sensori che permettono di comprendere in che stato si trova il sistema.

  Lo stato del sistema può variare anche per gli eventi esogeni.
- Il controller che invia azioni al sistema e legge le osservazioni del sistema per comprenedere in che stato si trova il sistema.
- Il planner che tramite una rappresentazione del sistema invia calcola il piano e lo invia al controllore che lo traduce in azioni da eseguire.

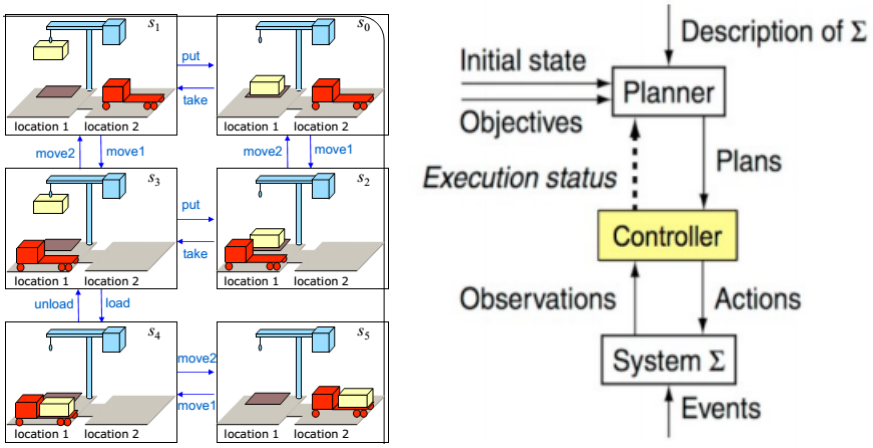


Figura 78: Sistema gru - base mobile.

Un esempio di pianificazione è il calcolo delle azioni per andare da uno stato iniziale a uno stato finale, ma ci sono infinite sequenze.

Quindi bisogna trovare la sequenza che rispetta qualche criterio di ottimalità.

Piani Il piano che il pianificatore invia al controllore può essere di due tipi:

- Classic plan: una sequenza di azioni da eseguire.
- Policy: una sequenza di azioni e stati in cui si dovrebbe essere per eseguire quella azione.

  La policy è più robusta, dato che se non ci si trova nello stato iniziale corretto si può comunque arrivare in fondo.

  Inoltre si può avere un feedback se l'azione è stata eseguita correttamente.

14.2 Pianificatore vs Scheduler

Pianificatore

- decide quali azioni eseguire per raggiungere l'obiettivo.
- può essere peggio che NP-completo, può essere indecidibile.

Scheduler Lo scheduling è come il planning, ma con delle considerazioni in più:

- decide quando e come eseguire un insieme di azioni
  - vincoli di tempo
  - vincoli sulle risorse
  - funzione obiettivo
- Di solito non è NP-completo

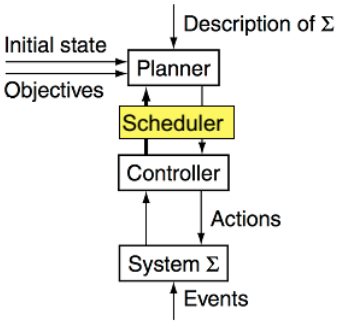


Figura 79: Sistema gru - base mobile con lo scheduler.

14.3 Domain specific

Sono pianificatori realizzati su «misura», in base all'ambiente/dominio in cui devono operare, quindi non sono pensati per funzionare bene in altri domini.

Sono i pianificatori migliori e più usati nel mondo, fino a oggi (e.g. Mars rover).

14.4 Domain indipendent

In teoria, un pianificatore domain-independent funziona in ogni dominio perchè ha una conscenza generale (escluse ovviemente le azioni base che operano nel dominio in questione).

In pratica, non è fattibili realizzare pianificatori simili, perchè bisogna fare delle assunzioni restrittive a priori per un determinato insieme di domini.

Assunzioni restrittive

- ***A0***: sistema finito: numero finito di stati, azioni ed eventi esogeni.
- ***A1***: sistema interamente osservabile: il controllore conosce sempre lo stato corrente del sistema Σ.
- ***A2***: deterministico: ogni azione ha un solo significato.
- ***A3***: statico (no eventi esogeni): non si hanno cambiamenti nel sistema, tranne quelli portati dalle azioni del robot.
- ***A4***: obbiettivi raggiungibili: insieme di stati obiettivo S_g.
- ***A5***: piano sequenziale: il piano è lineare e consiste in una sequenza di azioni $(a_1, a_2, \ldots, a_n)$.
- ***A6***: tempo implicito: non si hanno tempistiche, il passaggio da uno stato all'altro è istantaneo.
- ***A7***: pianificazione off-line: il pianificatore ha informazioni sullo stato di esecuzione

In base alle ipotesi fatte si possono ottenere diversi algoritmi.

- Classical Planning: Tutte le ipotesi attive.
- Hierarchical Task Networks se si rilassa l'ipotesi di finitezza degli stati.
- Discrete Markov Decision Processes se si rilassa l'ipotesi di determinismo.
- Finite POMDP se si rilassa l'ipotesi di osservabilità e di determinismo.
- hybrid MDPs se si rilassa l'ipotesi di finitezza degli stati e di determinismo
- hybrid POMDPs se si rilassa il determinismo, l'osservabilità e la finitezza
- Se si rilassa anche la staticità del sistema allora si possono usare ancora le MPD e le POMDP con opportune modifiche.
- Se si rilassa la staticità e l'osservabilità allora ci si trova nel campo dei Imperfectinformation games.
- Se si rilassa l'ipotesi di piano sequenza e di tempo implicito si parla di temporal planning
- Se si rilassa la finitezza degli stati e del piano sequenziale allora si parla di scheduling,

14.4.1 Classical Planning

Tutte le ipotesi sono verificate e allora il piano corrisponde a trovare un percorso nel grafo dell'automa a stati finiti che porta dallo stato iniziale a quello finale.

Il problema è che il numero di stati cresce esponenzialmente con gli elementi del sistema e quindi diventa velocmente impraticabile, ma comunque è il sistema più usato tramite diversi algoritmi.

Gestione del grosso numero di stati Per evitare di avere un numero ingestibile di stati gli oggetti del sistema sono rappresentati tramite oggetti nella logica del prim'ordine o in una sua semplificazione senza funzioni.

In questo modo gli stati sono rappresentabili dall'insieme di predicati veri che esistono nel sistema, per esempio se ci sono degli oggetti che rappresentano delle locazioni L_i e degli oggetti che rappresentano delle casse C_i e un predicato $Pos\,(L_i, C_i)$ se nello stato corrente questo è vero allora si sa che la cassa C_i si trova nella locazione L_i.

In questo modo non c'è bisogno di rappresentare tutti i possibili stati, ma bisogna controllare se i vari predicati sono veri o falsi.

In questo sistema si possono anche definire delle azioni che hanno degli effetti sulla veridicità dei vari predicati e formalmente sono definiti da un tripla:

- Precondizioni che devono essere vere per eseguire l'azione rappresentate come una serie di predicati.
- Effetti ossia la variazione del valore dei vari predicati dopo aver fatto l'azione.
- Nome dell'azione seguita dall'elenco degli oggetti coinvolti tra parentesi

Per esempio l'azione $Sposta\,(C_1, L_1, L_2)$ che rappresenta lo spostamente di una cassa dal posto L_1 al posto L_2 ha:

- Precodinzione $Pos\,(L_1, C_1)$ ossia la cassa si trova nella posizione di partenza
- Effetto $Pos\,(L_1, C_1) = False$ e $Pos\,(L_2, C_1) = True$

Notazioni Questo sistema usa le seguenti notazioni:

- S^+ insieme di predicati veri nello stato S
- S^- insieme di predicati falsi nello stato S
- $Precond^+\,(a)$ condizioni che devono essere vere per eseguire l'azione a
- $Precond^-\,(a)$ condizioni che devono essere false per eseguire l'azione a
- $Effect^+\,(a)$ predicati che diventano veri dopo aver eseguito a
- $Effect^-\,(a)$ predicati che diventano falsi dopo aver eseguito a
- $\gamma\,(S, a)$ l'applicazione dell'azione a allo stato S

In questo modo si può eseguire un'azione facendo:

- Si controllano le precodinzioni facendo $Precond^+\,(a) \subseteq S^+$ e $Precond^-\,(a) \cap S^+ = \varnothing$
- Si esegue l'azione facendo $S^+ = (S^+ - Effect^-\,(a)) \cup Effect^+\,(a)$

Planning Problems Con questo sistema il planning problem è visto come:

- S_0^+ insieme dei predicati veri nello stato iniziale.
- g la formula in logica del primo ordine che deve essere vera per dire di aver raggiunto l'obbiettivo.
- O l'insieme degli operatori

e il piano è una sequenza di azioni tali che:

$$\begin{aligned} \gamma\,(S_0, a_1) &= S_1 \\ \gamma\,(S_1, a_2) &= S_2 \\ &\vdots \\ \gamma\,(S_{n-1}, a_n) &= S_n \end{aligned}$$

e in S_n g è verificata.

14.5 Pianificazione HTN (Hierarchical Task Networks)

Questo si applica se si rilassa l'ipotesi del numero finito di stati e di azioni e quindi ci si trova in un sistema modellizzato da variabili continue.

In questo metodo si cercano diversi metodi anche in domini differenti per eseguire l'obbiettivo e ognuno di questi metodi viene diviso in sottotask facilmente eseguibili, o a loro volta divisibili nello stesso modo.

Dopo di che il planner valuta cosa ha intorno, sceglie il metodo migliore in base a dei vincoli che si aggiungono ai vari metodi.

14.5.1 Simple Task Network (STN) Planning

E' un caso particolare dell'HTN in cui sono presenti operatori e stati in modo equivalente al classical planning a cui si aggiungono:

- Task ossia un'azione che il robot può eseguire e che può essere di due tipi:
  - Primitiva, ossia si esegue facilmente.
  - Non primitiva, ossia va divisa in altri sottotask da eseguire in sequenza.
- Metodi ossia una soluzione a un task non primitivo che è composto da una quadrupla:
  - Il nome del metodo seguito da tutte le variabili coinvolte tra parentesi.
  - Il task che risolve
  - Le precodizioni che devono essere vere per eseguire il task
  - Una sequenza di subtask che risolvono il task più complesso. Questo listapuò essere anche solo parzialmente ordinata ossia ci possono essere subtask interscambiabili.

Tramite questo sistema è possibile lasciare al planner la scelta dei metodi da usare per risolvere un task. e la soluzione è una qualsisi sequenza di task primiti eseguibili.

14.6 Markov Decision Processes

Questo sistema si utilizza quando le azioni hanno esito incerto e quindi l'automa a stati finiti del sistema si complica aggiungendo il termine $P_a(s_1|s_2)$ che è la probabilità di andare nello stato s_1 partendo dallo stato s_2 usando l'azione a e si ha sempre che:

$$\sum_{s \in S} P_a(s|s_i) = 1$$

ossia applicando una azione a dallo stato s_i si arriva sicuramente da qualche parte.

Uso delle policy Dato che l'esito delle azioni è incerto bisogna usare le policy in modo che il controllore sappia sempre dove l'azione lo ha portato ed esegue sempre l'azione giusta.

Data una determinata policy questa può crare un determinato numero di percorsi ai quali si può associare una probabilità che questo avvenga e, basandosi su queste probabilità si può calcolare qual è la probabilità che quella policy porti al goal. Quindi avere un criterio di scelta della policy.

Pesi delle azioni Il sistema di prima considera tutte le azioni uguali, ma potrebbe darsi che in realtà alcune azioni costino di più in termini di risorse e quindi si può aggiungere al sistema la funzione di costo $C(s, a)$ che dice quanto costa fare l'azione a dallo stato s.

Reward degli stati Magari passare in determinati stati è preferibile e quindi si può aggiungere la funzione $R(s)$ che dice quanto è bello passare da quello stato.

Utilizzando queste due funzioni si può assegnare un'altro punteggio ai vari percorsi e quindi scegliere qualcosa di diverso.

Un modo per applicare questo concetto è usare la funzione di discountà V ossia dato un percorso h e una policy π:

$$V(h|\pi) = \sum_{i \geq 0} \gamma^i \cdot (R(s_i) - C(s_i, \pi(s_i)))$$

con $0 \leq \gamma < 1$ che è un modo per scontare i costi delle cose lontane nel percorso, utilizzando questo funzione si può valutare la policy facendo:

$$E(\pi) = \sum_h P(h|\pi) \cdot V(h|pi)$$

ossia si fa la media dei valori dei percorsi pesata con la probabilità che questi avvengano.

Policy ottima Una policy si dice ottima se:

$$E(\pi) \leq E(\pi') \ \forall \pi'$$

e si dice ottima ovunque anche se si partisse da altri stati.

14.7 Partially observable Markov Decision Process (POMDP)

In questo planner il sistema non è in grado di osservare con chiarezza l'esito delle azioni e quindi al sistema visto prima bisogna aggiungere:

$$P_a(o|s)$$

ossia la probabilità di osservare o dopo aver eseguito a nello stato s e, dato che bisogna sempre osservare qualcosa:

$$\sum_{o \in O} P_a(o|s_i) = 1$$

questi sistemi sono difficili da risolvere

Comprendere in che stato ci si trova Per comprendere in che stato ci si trova bisogna applicare una azione specifica per vedere l'osservazione.

Quindi stabilire tramite l'osservazione in che stato ci si trovava prima, e quindi ora, con maggiore probabilità.

Per esempio se si ha a disposizione una azione a che non cambia mai stato e una osservazione o tale che:

$$P_a(o|s) = 0 \, \forall s' \neq s$$

$$P_a(o|s) = 1$$

allora se si applica l'azione a si può stabilire se ci si trova nello stato s leggendo l'osservazione o.

In generale questo non avviene, ma si può comunque avere una informazione probabilistica da tutte le azioni a meno che:

$$P_a(o|s) = P_a(o|s')$$

per tutte le osservazioni e azioni e in questo caso gli stati s e s' sono irriconoscibili.

Belief States La probabilità di essere in un determinato stato data dall'informazione spiegata prima, si definisce belief.

Inoltre si definisce $b(s)$ la probabilità di essere nello stato s e si ha che:

$$\sum_{s \in S} b(s) = 1$$

e la policy diventa una funzione parziale che dato la belief dei vari stati dice che azione fare. Inoltre, dato che l'insieme delle possibile belief è infinito, pure la policy potrebbe essere infinita nonostante gli stati sono finiti e questo è un problema.

15 Meccanica dei manipolatori

Robot industriale

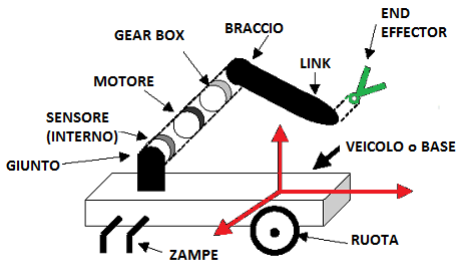


Figura 80: Parti dei robot.

è un manipolatore multifunzionale riprogrammabile, comandato automaticamente. Inoltre può essere ancorato a terra oppure ancorato su un carrello o rotaia in modo da poter riuscire a muoversi. I bracci in questa forma sono chiamati a cinematica aperta perchè è tutto libero di muoversi tranne il vincolo alla base.

Esistono a cinematica chiusa dove ci sono più basi che usano lo stesso end effector.

15.1 Giunti

Un braccio robotico è composto da dei link collegati tramite dei joint o giunti che possono essere di diverso tipo:

- Rotatori: in grado di fare ruotare un link rispetto a quell'altro (gomito).
- Traslatori: in grado di far traslare un link rispetto a quell'altro.

Ogni giunto è caratterizzato dal numero di gradi di libertà ossia dal numero di variabili che vanno definite per sapere con esattezza la posizione del giunto. Per ogni grado di libertà è necessario un motore.

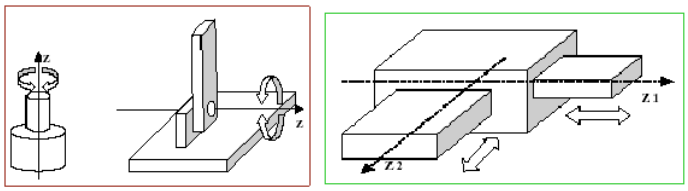


Figura 81: Giunti: giunto rotazionale e traslazione.

15.2 Link

I link sono caratterizzati dalla variazione di grado che comportano tra i due giunti a cui è collegato in particolare si dice parallelo se non porta variazione di grado e ortogonale se è $\frac{\pi}{2}$.

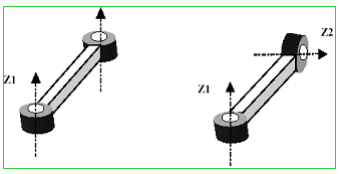


Figura 82: Link: parallelo sulla sinistra e ortogonale sulla destra.

15.3 Gradi di libertà

Un sistema composto da diversi giunti ha un numero di gradi libertà pari a

$$\mathbf{DOF} = 3 * (N–1)–2 * J1–J2$$

- **N** = numero dei link inclusa la terra;
- **J1** = numero dei giunti con 1 DOF;
- **J2** = numero dei giunti con 2 DOF;

Per operare in uno spazio tridimensionale servono 3 DOF e per ottenere qualsiasi orientamento altri 3 DOF. Quindi 3 DOF per il braccio e 3 DOF per il polso, **6 DOF**.

Se ci sono più gradi di libertà (**DOF** > 6), il sistema è sovradimensionato e può raggiungere la stessa posizione in più modi.

15.4 Spazio di lavoro

si intende l'insieme dei punti (posizioni) dello spazio che il robot può raggiungere con l'end effector. Si fa distinzione tra:

- spazio raggiungibile: dove la mano può essere posta con almeno un orientamento;
- spazio di destrezza: dove la mano può essere posta con ogni orientamento.

15.5 Esempi di bracci

15.5.1 Robot cartesiano (TTT)

Ha 3 giunti di traslazione, usato per assemblaggio in ambienti molto strutturati. Spazio di lavoro: parallelepipedo.

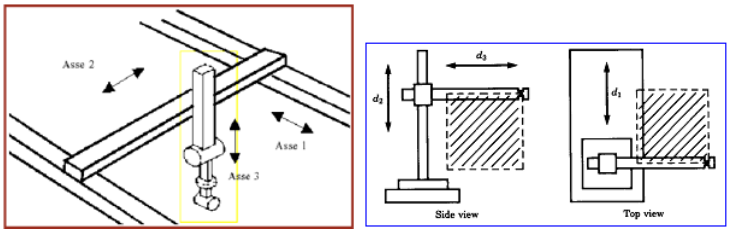


Figura 83: Robot cartesiano.

15.5.2 Robot cilindrico (RTT)

1 giunto di rotazione verticale e 2 di traslazione. Spazio di lavoro: cilindro. Meno libertà di movimento ma più semplice da realizzare.

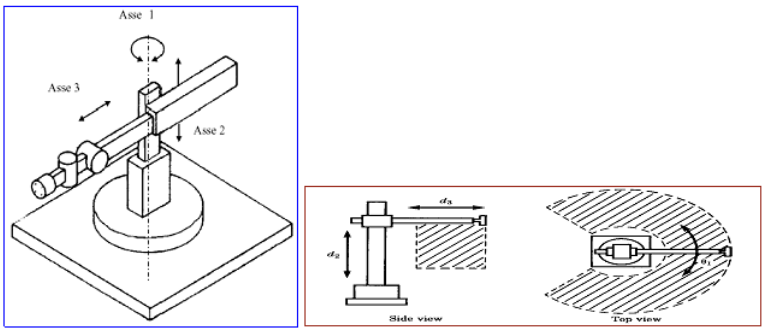


Figura 84: Robot cilindrico.

15.5.3 Robot sferico (RRT)

2 giunti di rotazione e 1 di traslazione. Spazio di lavoro: semisfera. Usato per operazioni di saldatura.

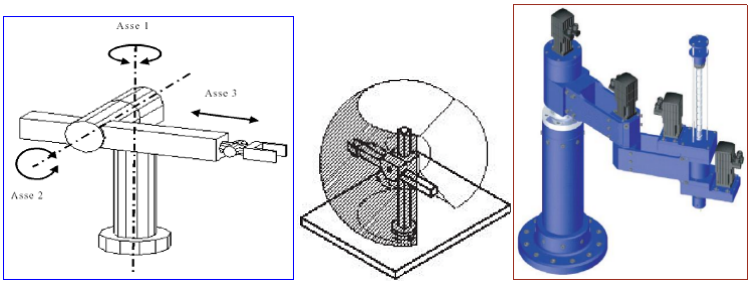


Figura 85: Robot cartesiano.

15.5.4 Robot articolato (RRR)

3 giunti di rotazione. Spazio di lavoro: parte di semisfera. Simile alla struttura del braccio umano, agile e facile da costruire.

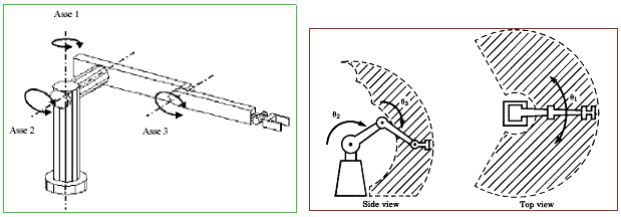


Figura 86: Robot articolato.

15.6 Polso di Robot

Mentre il braccio si occupa di raggiungere tutte le posizioni, il polso si occupa di raggiungere tutte le rotazioni possibili.

I polsi dei robot sono caratterizzati da:

- Accuratezza: l'errore che il polso commette sulla posizione alla fine del modo
- Ripetibilità: la variazione dell'errore che si ha tra un comando e lo stesso ripetuto
- Risoluzione: la distanza minima percepita dai sensori interni
- Massimo payload: il massimo peso che può portare il polso
- Massima velocità: la massima velocità che il braccio può raggiungere
- Tempo di ciclo: il tempo necessario per eseguire una azione di pick and place. In un robot di ottime prestazioni è inferiore al secondo.

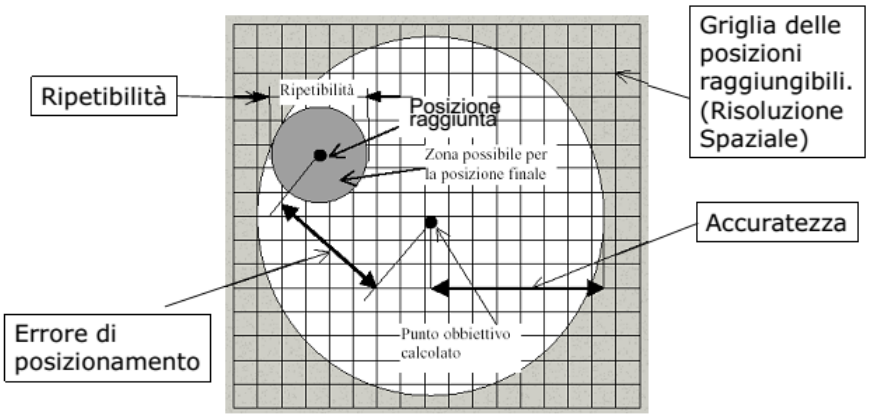


Figura 87: Accuratezza e ripetibilità.

La griglia rappresenta l'insieme delle posizioni raggiungibili cioè la risoluzione spaziale.

La posizione raggiunta è quella presa sul campo che viene memorizzata come variabili di giunto.

L'errore di posizionamento dipende dall'accuratezza del modello cinematico.

L'accuratezza del modello cinematico dipende da parametri geometrici (tolleranze) cedevolezza, etc.

E' più facile costruire robot ripetibili piuttosto che robot accurati.

16 Cinematica dei manipolatori

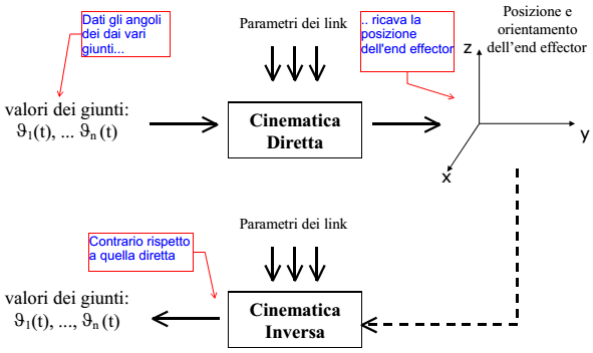


Figura 88: Cinematica diretta e inversa.

Definizioni

- La variabile di giunto è un parametro che permette di caratterizzare perfettamente lo stato di un giunto, e sono dei vettori n-dimensionali dove n è il numero di gradi di libertà del giunto.
- La cinematica diretta è il calcolo della posizione dell'end effector in base al valore delle varie variabili di giunto. Solitamente questo problema è caratterizzato da un sistema a n equazioni e $n$incognite, dove n è il numero di gradi di libertà del braccio.
- La cinematica inversa è il calcolo del valore delle variabili di giunto a partire dalla posizione dell'end effector. Solitamente questo si ottiene cercando le formule inverse di quelle della cinematica diretta, ma questo comporta tanti problemi:
  - Non sempre è possibile e facile farlo.
  - A volte esistono più soluzioni, ossia ci sono diverse configurazione dei giunti che portano alla stessa posizione come in figura 89.

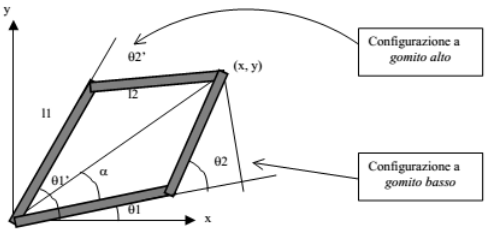


Figura 89: Cinematica inversa.

16.1 Cinematica diretta

Catena cinematica aperta é una catena di link-joint in cui vi è solo una sequenza che connette l'end effector a terra. In questo caso la cinematica diretta si trova tramite il metodo di Denavit Hartemberg.

16.1.1 Denavit Hartemberg

Il metodo di Denavit Hartember (DH) è utilizzato per rappresentare delle catene cinematiche aperte con un formalismo e fa in modo che una trasformazione geometrica possa essere rappresentata nello spazio euclideo tridimensionale con il numero minimo di parametri.

DH analizza un link alla volta, conoscendo la posizione del giunto J_i e le variabili di giunto di J_{i+1} si vuole trovare la posizione di J_{i+1}, iterativamente fino ad arrivare all'end effector.

Il metodo di Hartemberg calcola la matrice di rototraslazione che bisogna applicare alla posizione P_i del giunto i per avere la posizione P_{i+1} del giunto $i+1$ dove gli assi del sistema di riferimento sono assegnati con la seguente convenzione:

- l'asse Z_i corrisponde con l'asse del giunto
- l'asse X_i corrisponde alla linea che collega il giunti i a quello $i+1$
- l'asse Y si sceglie di conseguenza.

e per farlo la suddivide in 4 contributi:

- Una rotazione intorno all'asse x di un angolo α_i in modo che i due assi Z siano paralleli.
- Una traslazione lungo l'asse X delle lunghezza a_i in modo da avere la coordinata Z concidente con quella del giunto $i+1$
- Una rotazione intorno all'asse Z dell'angolo θ_{i+1} in modo da non modificare l'asse Z, che è giusto dopo la prima rotazione, e rendere parallelo anche l'asse X.
- Una traslazione sull'asse Z di una lunghezza d_{i+1} in modo da allineare il riferimento con quello del giunto $i+1$.

Se il giunto è di traslazione la variabile di giunti è θ_{i+1} e $d_{i+1} = 0$ mentre se è di traslazione $\theta_{i+1} = 0$ e d_{i+1}.

I parametri a_i e α_i invece sono i parametri costanti del link ossia la sua lunghezza e la sua variazione di angolo.

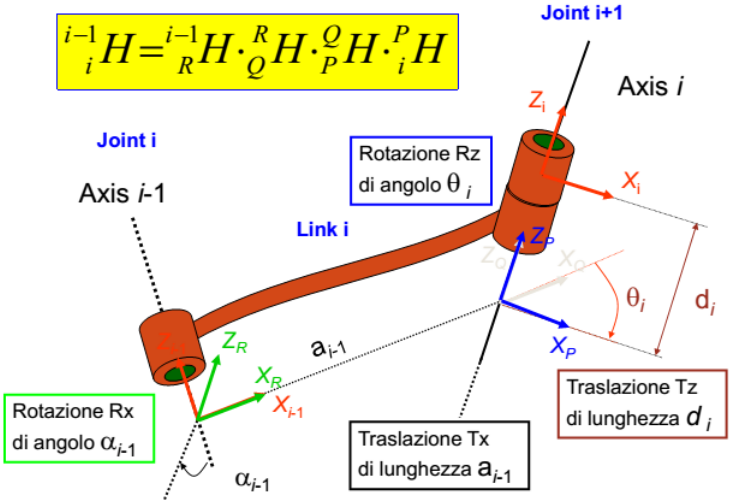


Figura 90: Descrizione di due giunti consecutivi con Denavit Hartemberg

Moltiplicando le 4 matrici di rototraslazione si ottiene la matrice di rototraslazione che permette di ricavare la posizione del giunto successivo. Chiamata

$$H_i^{i-1}$$

infine si può ricavare la posizione dell'end-effector applicando tutte queste matrici di rototraslazione in sequenza della posizione della terra:

$$\begin{aligned}
H &= \prod_i H_i^{i-1} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\alpha_{i-1}) & -s(\alpha_{i-1}) & 0 \\ 0 & s(\alpha_{i-1}) & c(\alpha_{i-1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & \alpha_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & 0 \\ s(\theta_i) & c(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & \alpha_{i-1} \\ s(\theta_i) \cdot c(\alpha_{i-1}) & c(\theta_i) \cdot c(\alpha_{i-1}) & -s(\alpha_{i-1}) & -d_i \cdot s(\alpha_{i-1}) \\ s(\theta_i) \cdot s(\alpha_{i-1}) & c(\theta_i) \cdot s(\alpha_{i-1}) & c(\alpha_{i-1}) & d_i \cdot c(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Quindi si è ottenuto una matrice H che dipende dalle varie variabili di giunto, θ_i o d_i tale che:

$$P_6 = H \cdot P_0$$

Ultimo giunto dato che l'ultimo giunto non ha giunto successivo se si vuole trovare la posizione della mano bisogna stabilire quale sia il sistema di riferimento della mano e solitamente si mette il sistema tra le dita, Y in modo che passi tra tutte le dita e Z in modo che esca dalle dita e X di conseguenza.

Robot ridondanti Se ci sono più modi per raggiungere la stessa posizione si dice che la posizione è ridondante e se ci sono inifinite posizioni allora si dice che il punto è degenere e bisogna costruire robot in modo da evitare questi punti infinitamente ridondanti.

Manipolatori con più di 6 DOF sono sempre ridondanti.

Algoritmo

1. Il robot in posizione di riposo. Si determina x_0, y_0, z_0.
2. Per $i = 1, \ldots, n-1$ si ripetono i passi ***3-6***.
3. Si stabilisce l'asse del giunto $i+1$ su Z_i.
4. Si posiziona l'origine O_i nel punto di intersezione fra Z_i e Z_{i-1} oppure nel punto di interezione del segmento di minima distanza fra gli assi stessi, con Z_i. Se Z_i e Z_{i-1} sono paralleli, si sceglie come segmento di minima distanza quello che è colineare con quello del link precedenti.
5. Si determina X_i sulla direzione del segmento di minima distanza oppure nella direzione del link, orientato verso la mano del robot.
6. Si determina l'asse Y_i con la regola della mano destra.
7. Si stabilisce il sistema di riferimento *n-esimo* nella mano.
8. Si trovano i parametri iterando per $i = 1, \ldots, n$ i passi da ***9 a 12***.
9. Si trova a_i, segmento di minima distanza fra Z_i e Z_{i-1}.
10. Si trova α_i angolo di rotazione fra Z_{i-1} e Z_i, attorno all'asse X_i.
11. Si trova d_i : distanza dall'origine del sistema $i-1$ all'intersezione del segmento di minima distanza fra Z_i e Z_{i-1}, con l'asse Z_{i-1}.
12. Si trova θ_i angolo di rotazione fra X_{i-1} e X_i attorno all'asse Z_{i-1}.

16.2 Cinematica inversa

Nella cinematica inversa conosco la posizione dei giunti del robot e la posizione dell'oggetto, devo calcolare le rototraslazioni necessarie per far arrivare il braccio sul punto di lavoro nell'oggetto. Quindi data la matrice delle rototraslazioni dei link

$$H = H_6^0 = H_1^0 H_2^1 H_3^2 H_4^3 H_5^4 H_6^5$$

e la posizione e l'orientamento dell'end effector è data dalla funzione H_6^0 in θ_i

$$H_6^0 = H(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$$

voglio calcolare i valori corrispondenti degli angoli dei giunti

$$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$$

Non è garantita né l'esistenza (punto di lavoro troppo lontano) né l'unicità (ci possono essere più modi per arrivare allo stesso punto di lavoro) della soluzione cinematica inversa.
La molteplicità delle soluzioni può essere utile per far evitare al robot degli ostacoli.

Il numero di soluzioni dipende dal numero di parametri $D - H \neq 0$, più ce ne sono a zero meno sono le soluzioni.

In genere se il robot ha 6 GDL allora ogni punto è raggiungibile ed esiste la soluzione, anche se magari non è unica.

Mentre se ne ha di meno è difficile determinare se tale soluzione esiste.

16.2.1 Molteplici soluzioni

Se esistono molteplici soluzioni allora serve un criterio per trovare la migliore. Per esempio minimizzando i movimenti che bisogna fare per raggiungerla, oppure si sceglie quella che evita gli ostacoli.

Avere più soluzioni può aiutare molto quando si vuole evitare gli ostacoli.

16.2.2 Metodi di soluzione

- Analitico, che sfrutta l'algebra delle matrici per risolvere il sistema non lineare, estremamente complesso.
- Geometrico, che sfrutta la geometria dei giunti e i vari teoremi geometrici, mandando a cagare la matrice.
- Iterativi che ripetono a ogni passo una azione per migliorare la soluzione e quindi avere una soluzione numerica.
- Calcolo in forma chiusa che garantisce la soluzione esatta.

16.2.3 Esempio

Date le coordinate x, y del polso del manipolatore calcolare i valori di θ_1 e θ_2 che permettono di raggiungere la posizione x, y

$$\begin{aligned}
\cos(\theta_2) &= \frac{(x^2 + y^2 - l_1^2 - l_2^2)}{2 * l_1 * l_2} \\
\theta_2 &= \arccos\left[\frac{(x^2 + y^2 - l_1^2 - l_2^2)}{2 * l_1 * l_2}\right]
\end{aligned}$$

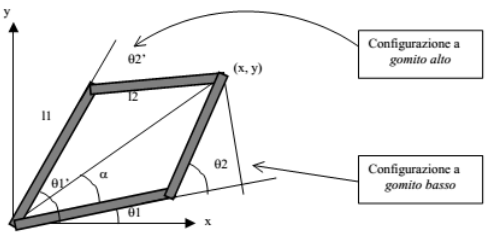


Figura 91: Cinematica inversa.

Il calcolo precedente porta a due soluzioni che differiscono per segno.

Per trovare l'altro angolo si osserva che ponendo $\Delta\theta = \theta_1 + \alpha$ vale:

$$\begin{aligned}
\tan(\Delta\theta) &= \frac{x}{y} \\
\tan(\alpha) &= \frac{l_2 * S_2}{l_1 + l_2 * \cos(\theta_2)} \\
\theta_1 &= tan^{-1}\left(\frac{y}{x}\right) - tan^{-1}\left(\frac{l_2 * S_2}{l_1 + l_2 * cos(\theta_2)}\right)
\end{aligned}$$

16.2.4 Metodo di Paul

Si egualia la matrice di DH alla matrice T di rotazione che bisogna applicare per raggiungere quella determinata posizione.

Si cercano così delle coppie di elementi che contengono solo una variabile di giunto in modo da ottenere una equazione risolvibile e risolvere almeno una variabile. Ripetendo questi passaggi fino alla fine si può risolvere il sistema e trovare la soluzione.

Se non si trova questa coppia allora si può trovare dei sistemi equivalenti facilmente, infatti si ha:

$$\begin{aligned} T &= H \\ T &= H_1 \cdot H_2 \cdot \dots \cdot H_N \\ H_1^{-1} \cdot T &= H_2 \cdot \dots \cdot H_N \end{aligned}$$

dove H_i sono le matrici di rototraslazione dei vari giunti.

16.2.5 Sostituzione di variabile

Per semplificare i conti solitamente si effettua una sostituzione di variabile:

$$u_i = \tan \frac{\theta_i}{2}$$

infatti solitamente sono tutte variabili dentro dei coseni e dei seni e si può dimostrare che:

$$\cos \theta_i = \frac{1 - u_i^2}{1 + u_i^2}$$

$$\sin \theta_i = \frac{2 \cdot u}{1 + u_i^2}$$

in modo da rendere le equazioni polinomiali invece che trigonometriche.

16.2.6 Caso particolare con 3 giunti rotatori e un polso a 3 gradi di libertà

In questo caso si hanno 3 variabili di giunto del braccio che si possono trovare utilizzando il metodo di Paul e paragonandole ai termini dell'ultima colonna della matrice di rototraslazione, dove c'è la traslazione.

Infatti la posizione dipende solamente dal braccio e non dal polso.

Una volta trovate le prime 3 variabili di giunto e ricordando che:

$$\begin{aligned} H &= H_1(\theta_1, \theta_2, \theta_3) \cdot H_2(\theta_4, \theta_5, \theta_6) = T \\ T \cdot H_1^{-1} &= H_2 \end{aligned}$$

Dato che H_2 è una matrice di pura rotazione si può applicare il metodo di Paul.

16.3 Cinematica differenziale

La cinematica differenziale si occupa di trovare la relazione tra la velocità dell'end effector e quella dei giunti.

Date le relazioni di cinematica diretta si possono derivare nel tempo per ottenere quelle in velocità e così facendo si ottiene una nuova matrice detta Jacobiana J tale che:

$$\dot{P} = J(\theta) \cdot \dot{\theta}$$

dove θ sono le variabili di piunto e P la posizione del giunto. La Jacobiano J dipende dalle variabili di giunto θ e quindi bisogna ricalcolarla per tutte le posizioni dei giunti.

16.3.1 Esempio

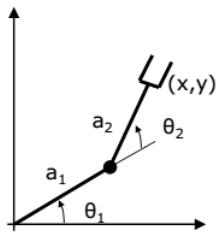


Figura 92: Esempio di robot RR.

La posizione dell'end-effector viene calcolata nel seguente modo

$$\begin{cases} x = a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \\ x = a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \end{cases}$$

derivando nel tempo si ottiene

$$\begin{cases} \dot{x} = -a_1 \sin(\theta_1)\dot{\theta}_1 - a_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \\ \dot{y} = a_1 \cos(\theta_1)\dot{\theta}_1 + a_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \end{cases}$$

successivamente rappresendo in forma matrciale si ottiene la Jacobiana

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -a_1 \cos(\theta_1) - a_2 \cos(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) \\ a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) & a_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \mathbf{J} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

La Jacobiana viene calcolata in funzione della posizione dei giunti del robot, quindi al variare delle loro posizioni la matrice va ricalcolata.

16.3.2 Jacobiana per piccole variazioni

Dato che:

$$\begin{aligned} \frac{d}{dt}P &= J(\theta) \cdot \frac{d}{dt}\theta \\ J(\theta) &= \frac{d}{dt}P \cdot \left(\frac{d}{dt}\theta\right)^{-1} \\ J(\theta) &= \frac{dP}{dt} \cdot \frac{dt}{d\theta} \\ J(\theta) &= \frac{dP}{d\theta} \end{aligned}$$

e quindi si ha che:

$$dP = J(\theta) \cdot d\theta$$

e quindi se il robot si trova in una determinata posizione e si vuole sapere come varia la posizione con una piccola variazione delle variabili di giunto si può usare questa moltiplicazione.

16.3.3 Piccole rototraslazioni

Se si vuole applicare una matrice di rototraslazione per angoli piccoli allora si possono applicare le seguenti approssimazioni:

$$\begin{aligned} \sin(d\theta) &= d\theta \\ \cos(d\theta) &= 1 \end{aligned}$$

questo semplifica molto le matrici inoltre considerando nulli i termini quadrati e quindi non conta l'ordine di moltiplicazione.

Chiamando la nuova matrice di rototraslazione si ottiene:

$$H_{new} = R \cdot H_{old}$$

e quindi la variazione data vale:

$$\begin{aligned} dH &= H_{new} - H_{Old} \\ &= R \cdot H_{old} - H_{old} \\ &= (R - I) \cdot H_{old} = \\ &= \Delta \cdot H_{old} \\ &= \begin{bmatrix} 0 & -\delta\varphi & \delta\theta & \delta x \\ \delta\varphi & 0 & -\delta\psi & \delta y \\ -\delta\theta & \delta\psi & 0 & \delta z \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot H_{old} \end{aligned}$$

dato che la matrice R ha solo 1 sulla diagonale si ha che la matrice Δ ha solo 0 sulla diagonale.

Quindi data una posizione si può ricavare cosa succede se si esegue una piccola rototraslazione calcolando la matrice Δ, che è una matrice molto semplice e facilmente calcolabile.

16.3.4 Relazione tra Δ e J

- Sono date le matrici di rototraslazione tra il polso del robot e la base del robot: $H = H_6^0 = H_1^0 H_2^1 H_3^2 H_4^3 H_5^4 H_6^5 = H(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$
- Si calcola la Jacobiana **J** associata a H
- E' dato il vettore delle variazioni dei giunti $D_\theta = [\delta\theta_1, \delta\theta_2, \delta\theta_3, \delta\theta_4, \delta\theta_5, \delta\theta_6]$
- Si calcola la Jacobiana **J,** il vettore delle variazioni di posizione e orientamento del polso del robot: $D = \mathbf{J} \cdot D_\theta = [\partial x, \partial y, \partial z, \partial\psi, \partial\theta, \partial\varphi]$
- Si calcola l'operatore differenziale: $dH = (H_{\delta R, \delta T} - I) \cdot H_{old} \longrightarrow dH = \Delta \cdot H_{old}$
- Si calcola la nuova matrice di rototraslazione tra il polso del robot e la base del robot: $H_{new} = H_{old} + \triangle \cdot H_{old} = H_{old} + dH$

16.3.5 Cinematica differenziale inversa - Jacobiana inversa

Se il numero di gradi di libertà dell'end-effector (max 6) è uguale al numero di giunti del robot, matrice Jacobiana è quadrata e allora può essere invertibile. Se **J** è invertibile allora

$$\dot{\theta} = \mathbf{J}^{-1} \cdot \dot{x}$$

.

$$\mathbf{J}^{-1} = \frac{1}{a_1 a_2 sin\,(\theta_2)} \cdot \begin{bmatrix} a_2 cos\,(\theta_1 + \theta_2) & a_2 sin\,(\theta_1 + \theta_2) \\ -a_1 cos\,(\theta_1) - a_2 sin\,(\theta_1 + \theta_2) & -a_1 cos\,(\theta_1) - a_2 cos\,(\theta_1 + \theta_2) \end{bmatrix}$$

Se il controllore si trova in un punto di singolarità non sa cosa fare, quindi si blocca e potrebbe danneggiare il robot. Bisogna allora evitare che il controllore porti il robot in un punto di ***singolarità***.

$$\begin{aligned} det\,(\mathbf{J}) &= a_1 a_2 sin\,(\theta_2) \\ det\,(\mathbf{J}) &= 0 \;\; se\;\theta_2 = k\pi \end{aligned}$$

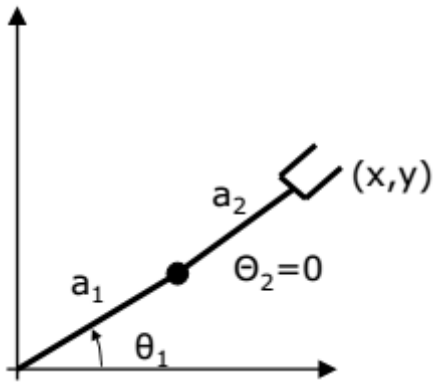


Figura 93: Esempio singolarità di robot RR.

Se per esempio il braccio è completamente steso, ci sono delle direzioni di movimento impossibili (e.g. andare dritto in una direzione). Allora non si riesce a calcolare alcune variazioni.

16.4 Metodi iterativi per la Cinematica Inversa

Notazioni Sia e la posizione dell'end-effector e θ il valore delle variabili dei giunti allora la cinematica diretta è data da:

$$e = f(\theta)$$

e la cinematica inversa:

$$\theta = f^{-1}(e)$$

ossia trovare una funzione che data la posizione dell'end effector mi dice come devono essere messi i giunti.

16.4.1 Metodo del gradiente

Alla fine è solamente un problema di calcolo degli zeri:

$$e - f(\theta) = 0$$

e quindi si può procedere con i metodi di newton e in particolare quello del gradiente in cui si sceglie la variazione su una coordinata come l'inverso della derivata parziale di quella coordinata.

Per esempio si consideri un braccio con un solo link e un solo giunto rotatorio con variabile di giunto q allora la posizione dell'end effector vale:

$$\begin{aligned} x &= l \cdot \cos q \\ y &= l \cdot \sin q \end{aligned}$$

allora si ha che:

$$\frac{dx}{dq} = -l \cdot \sin q$$

e quindi:

$$dq = -\frac{1}{l \cdot \sin q} \cdot dx$$

che è la variazione sulla x che si esegue a ogni iterazione e quindi l'algoritmo diventa:

- Si calcola $x_i = l \cdot \cos q_i$ ossia la posizione al passo i
- Si ricava l'errore e lo si moltiplica per una costante che va calibrata per avere un compromesso tra stabilità e velocità $dx = \alpha \cdot (x_g - x_i)$
- si ricava la variazione su q ottenendo $dq = -\frac{1}{l \cdot \sin q} \cdot dx$
- si calcola $q_{i+1} = q_i + dq$

16.4.2 Jacobian Inverse Kinematics

Come visto prima:

$$de = J \cdot d\theta$$

e quindi se si vuole applicare questo sistema per la cinematica inversa:

$$d\theta = J^{-1} \cdot de$$

e si può usare questo come passo iterativo:

- Si calcola il jacobiano J e lo si inverte J^{-1} nella posizione θ_i
- Si calcola l'errore $\Delta e = \beta \cdot (g - e_i)$ dove β è una costante.
- Si ricava $\theta_{i+1} = \theta_i + J^{-1} \cdot \Delta e$
- Si calcola e_{i+1}

bha... anche questo è un metodo di newton per il calcolo dello zero, anzi questo è quello vero.

16.4.3 Inverting the Jacobian Matrix

Invertire la matrice jacobiana porta con se dei problemi:

- Potrebbe non essere quadrata
- Potrebbe essere singolare in qualche punto
- Potrebbero esserci più soluzioni
  Questo avviene se il sistema è Underconstrained ossia se ci sono più gradi di libertà nei giunti che nell'end effector, ma in questi casi una soluzione la si trova comunque, ma trovare la migliore è ostico.
- Potrebbero non esserci soluzioni
  Questo avviene se il sistema è Overconstrained ossia ci sono più gradi di libertà nell'end effector che nei giunti, ma comunque ci si vuole avvicinare il più possibile e in pratica non avverà quasi mai.

Supponendo che il sistema non sia particolare(ossia gli assi sono bene messi) e quindi ha una sola soluzione bisogna risolvere la questione della quadratezza e lo si fa con la pseudoinversa:

$$J^{\dagger} = \left(J \cdot J^T\right)^{-1} \cdot J^T$$

oppure un secondo metodo è usare semplicemente la sua matrice trasposta che stranamente funziona abbastanza bene.

Quando fermarsi Ci si ferma quando:

- La precisione è abbastanza
- è finito il tempo a disposizione

Minimi locali A volte l'algoritmo si ferma nei minimi locali, come tutti gli algoritmi di newton, e in questi casi:

- Si accetta la soluzione e pota
- si cambia algoritmo
- si sposta la posizione inizale dell'algoritmo a caso e si spera di non finire nel minimo locale

16.4.4 Problemi della Inverse Kinematics

Joint Limits Tutto quello visto si basa sull'ipotesi che i joint sono derivabili ovunque, ma in realtà hanno dei limiti sui bordi della zona di lavoro. Per risolve il problema:

- Ogni volta che si calcola una posizione la si pone restringe nel possibile
- Se la derivata si trova sul bordo la si mette a una valore fisso in base ad altre assunzioni

Repeatability La soluzione generata non è sempre la stessa anche avendo gli stessi input per via dei minimi locali e del fatto che potrebbero esserci più soluzioni ottime.

Se la si vuole ripetibile allora si può imporre di usare sempre la stessa condizione iniziale.

Più end effector Se si hanno più end effector una buona soluzione è considerarli come se composti da diverse catene e computarli separatamente.

17 Motion planning 2D

L'idea della pianificazione del movimento è quella di creare un percorso libero da ostacoli permettendo a un robot (corpo rigido e/o articolato), che si muove sul piano, di raggiungere l'obiettivo prefissato evitando di sbattere contro gli ostacoli (considerati statici).
Il robot conosce a priori alcune informazioni:

- la mappa dell'ambiente (in questo modo può sapere in anticipo dove sono posizionati gli ostacoli);
- la sua cinematica (conosce quanti DOF ha) e la sua geometria (URDF).
- posizione di partenza e posizione di arrivo (sul piano).

Il pianificatore genera continuamente delle sequenze di configurazioni libere da ostacoli, per raggiungere l'obiettivo.

17.1 Completezza del pianificatore

Un pianificatore di moto è completo se è sempre in grado di trovare un percorso libero da ostacoli, quando esso esiste, oppure di tornare un messaggio di non esistenza di tale percorso.

I seguenti metodi forniscono una completezza del pianificatore di moto: grafo delle visibilità, diagramma di Voronoi, decomposizione esatta, funzione di navigazione.
Si parla di *completezza debole* quando si ha

- **Pianificatore statisticamente completo**:
  - ritorna un percorso con alta probabilità se questo esiste, potrebbe non terminare se il percorso non esiste.
  - per completezza statistica, si intende che la probabilità che il pianificatore troverà il percorso da seguire, se questo esiste, sale allo scorrere del tempo (all'aumentare dell'esposizione dell'ambiente al robot).
- **Pianificatore a risoluzione completa**: discretizza lo spazio e ritorna un percorso quando ne esiste uno a quella risoluzione.

17.2 Spazio delle configurazioni

Il robot deve sapere dove **può** andare e dove **non può** andare quando si muove (se si muove) nell'ambiente.

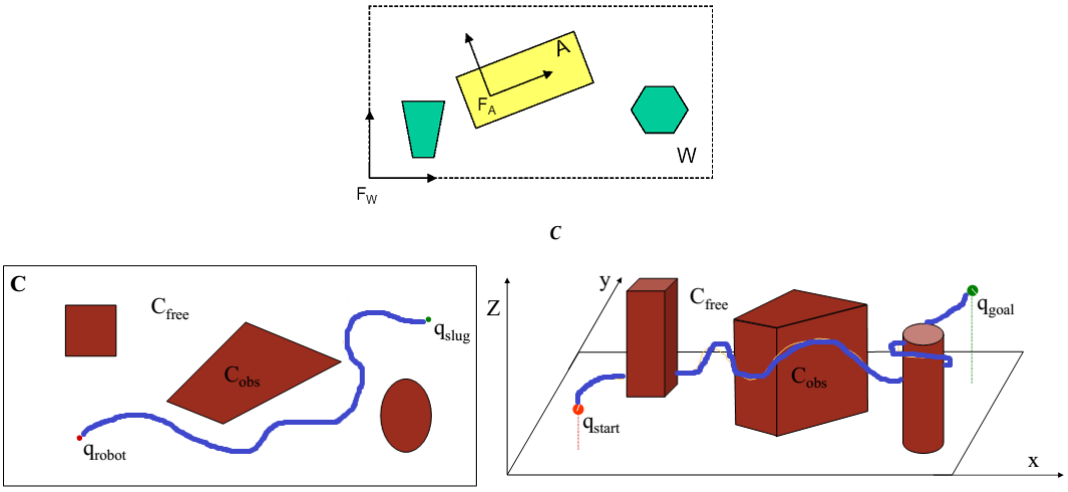


Figura 94: Spazio delle configurazioni.

Il robot consiste in un corpo A rigido o articolato (base mobile + rimorchio + altro) che si può muovere con un *frame* F_A nello spazio euclideo$W$ (può essere R^2 oppure R^3). Lo spazio W contiene degli ostacoli statici $B_1, B_2, \ldots, B_m$ che occupano determinate aree su cui il robot non può stare.

Sia q una configurazione di A con riferimento al *frame* fisso dell'ambiente F_W, definita da una posizione e un orientamento.

Si definisce quindi ***spazio delle configurazioni*** l'insieme di tutte le possibili configurazioni che può assumere il robot.

Per semplificare il concetto del frame, il robot viene individuato nello spazio euclideo da un singolo punto con tre parametri (x, y, θ).

Se il robot è circolare allora si possono usare solo 2 parametri (non serve l'orientamento θ, ma basta solamente (x, y)), per descriverlo.

17.2.1 Braccio robot - spazio delle configurazioni

Lo spazio di configurazione ha *due elementi* perchè il giunto ha *due DOF* (α e β) ed è uno spazio bidimensionale. Quindi lo spazio di configurazione non è uguale allo spazio cartesiano delle posizioni del robot.

L'ostacolo si sparge nello spazio di configurazione come in figura 95 perchè bisogna segnare le configurazioni (α, β) non praticabili (vietate), a causa della presenza dell'ostacolo.
Sugli assi dello spazio ci sono i gradi che possono raggiungere i due angoli. Se il robot avesse avuto sei DOF allora lo spazio delle configurazioni sarebbe stato a sei dimensioni.

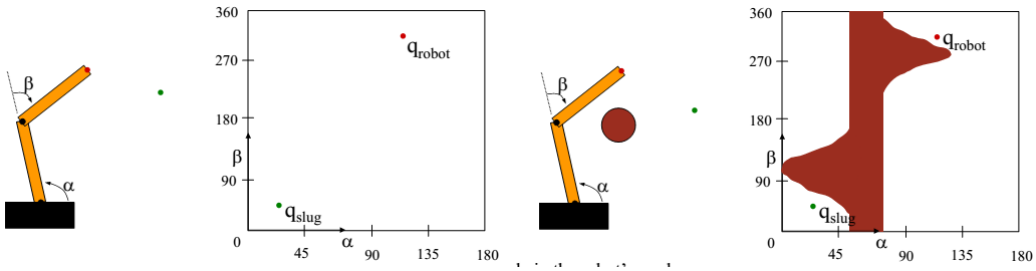


Figura 95: Braccio articolato con 2 DOF con e senza ostacolo.

Per ottenere la configurazione del piano cartesiano, bisogna generare tutte le possibili configurazioni di (α, β) e successivamente verificarle affinchè il robot non collida conl'ostacolo. SOLUZIONE IMPRATICABILE

17.2.2 Il robot non è un punto

Problema Come si può rappresentare il robot tale da poter rappresentare il suo movimento e ottenere in modo semplice (α, β)?

Idea Il robot è definito da un cerchio e gli ostacoli sono ben definiti. Allora si procede estendendo l'ingombro degli ostacoli tanto quanto è il raggio del robot e si riduce ovviamente la dimensione del robot tanto quanto è stato l'aumento degli ostacoli.

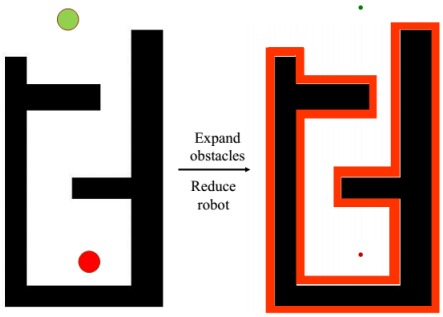


Figura 96: Espansione dei bordi l'ostacolo

17.3 Minkowski's sum

Data la forma irregolare del frame del robot e quella di un'ostacolo, si fa muovere il robot lungo il perimetro del'ostacolo e si allarga l'ingombro di quets'ultimo.

A volte tale procedimento è sfavorevole perchè taglia fuori alcune aree della regione libera dato che l'ostacolo si espande troppo.

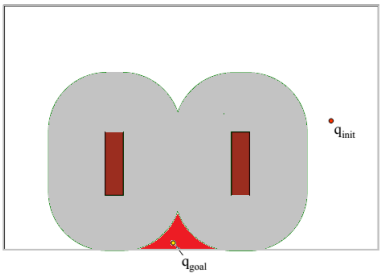


Figura 97: Espansione degli ostacoli ed «errori».

17.4 Estensione del problema

Ipotizzando che gli ostacoli non siano statici, ma dinamini, e sapendo che esiste dell'incertezza sulla posizione e configurazione attuale del robot. Si arriva ad estendere ulteriormente gli ostacoli anche di un valore corrispondente all'incertezza.
L'idea quindi è quella di passare per i seguenti punti:

1. Rappresentazione continua del problema: spazio cartesiano, configurazione del robot (DOF), vincoli.
2. Discretizzazione: consiste in un grafo in cui i nodi sono le celle e gli archi i percorsi che le collegano.
3. Ricerca in un grafo.

17.5 Approcci alla pianificazione dei percorsi

Siano definiti i seguenti concetti:

- percorso libero: tutti i punti del percorso appartengono allo spazio libero;
- percorso semi - libero: alcuni punti del percorso appartengono agli ostacoli.

17.5.1 Global planning

I metodi globali utlizzano hanno informazioni circa l'ambiente che circonda il robot. Quindi l'obiettivo è quello di trovare un percorso libero da ostacoli, riducendo l'ingombro del robot a un singolo punto e aumentando di conseguenza quella degli ostacoli.

17.5.2 Local planning

Questa tipologia di pianificazione utilizza metodi che non risolvono l'intero problema, ma utilizzano le informazioni raccolte dal robo fino a quel momento per effettuare il passo successivo.

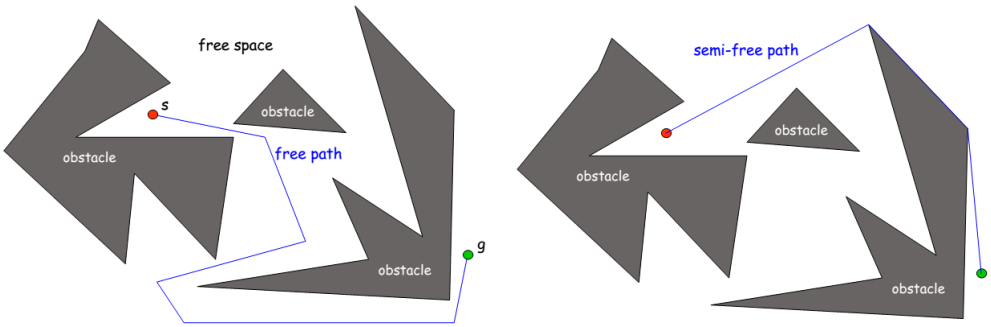
17.5.3 Algoritmi pratici

Il pianificatore di moto è molto oneroso in termini di risorse e richiede tanto tempo per processare i dati e trovare un percorso libero da ostacoli. In generale il pianificatore di moto dà sempre un percorso libero da ostacoli in output, se questo esiste, altrimenti segnala la sua non esistenza.

Gli algoritmi si suddividono in due tipi

- teorici: si impegnano a trovare una soluzione completa con una complessità minima, ma sono difficili da implementare e non sono robusti.

- euristici: si impegnano a essere efficienti, ma non sono affidabili in termini di completezza, performance. Sono quelli più usati.



(a) Percorso libero da ostacoli. (b) Percorso semi - libero da ostacoli.

Figura 98: Percorsi liberi e semi - liberi da ostacoli.

17.6 Decomposizione in celle

L'algoritmo suddivide lo spazio libero in sempli celle e rappresenta la connessione delle celle con degli grafi che hanno come nodi le celle e archi l'adiacenza delle celle.

17.6.1 Decomposizione esatta

La suddivisione dello spazio libero in celle è esatta. Quindi non ci sono sovrapposizioni delle celle nell'insieme delle celle: **decomposizione trapezoidale**.

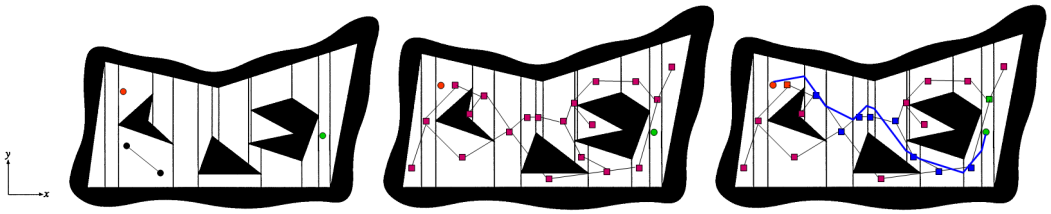


Figura 99: Decomposizione esatta: trapezoidale.

A ogni cella si associa un punto che il robot raggiunge se si vuole che raggiunga tale cella. *L'algoritmo è completo ma non è ottimo.*

17.6.2 Decomposizione approssimata

Lo spazio libero è rappresentato da un insieme di celle che non si sovrappongono tra di loro: **quadtree**, octree, 2n-tree.
Utilizzando il metodo quad-tree si suddivide lo spazio libero in quattro aree e successivamente ogni area viene suddivisa in tante sotto celle. Alla fine dell'algoritmo si ottiene un'albero di celle.

L'algoritmo si ferma quando la cella è piccola a piacere, oppure quando l'albero è profondo a piacere.

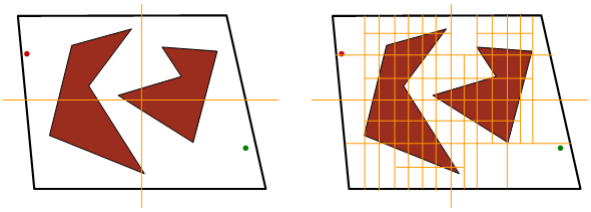


Figura 100: Decomposizione approssimata: quad-tree.

17.7 Campi potenziali

Il robot non ha una rappresentazione geometrica di tutto l'ambiente, ma ha un comportamento reattivo.

Il robot non fa altro che impegnarsi a evitare gli ostacoli per raggiungere il goal usando i sensori a bordo.

Per ogni tipologia di ostacoli, si ha un diverso comportamento, dato che sono modellati in campi di forze: (a) uniformi, (b) perpendicolari, (c) attrattive, (d) repulsive e (e) tangenti.

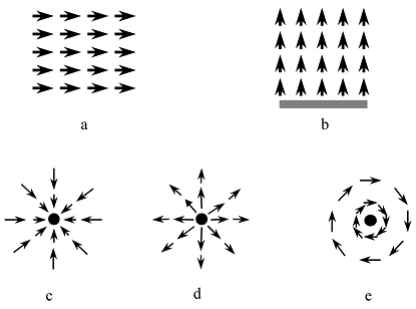


Figura 101: Diversi tipi di campi di forza degli ostacoli.

I campi di forza si possono combinare insieme per ottenere diverse traiettorie.

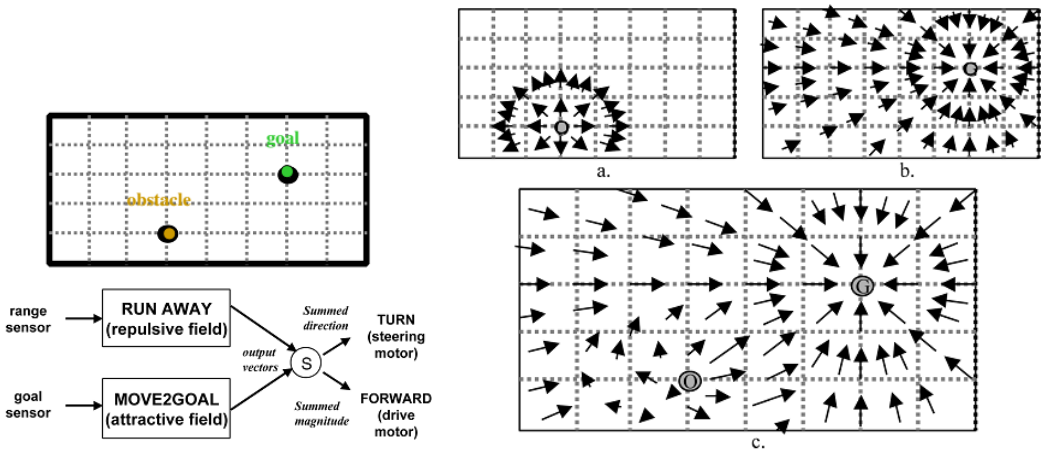


Figura 102: Combinazione campi di forza.

Il robot immerso nei campi di forza dell'ambiente può essere atratto da una serie di forze che corrispondono alla forza del percorso che deve seguire.

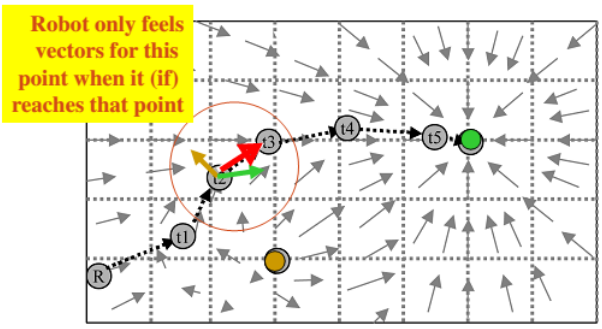


Figura 103: Combinazione campi di forza.

Vantaggi dei campi di forza

- facili da visualizzare

- facile implementazione software
- i campi possono essere parametrizzati
- guadagni e meccanismi fissi

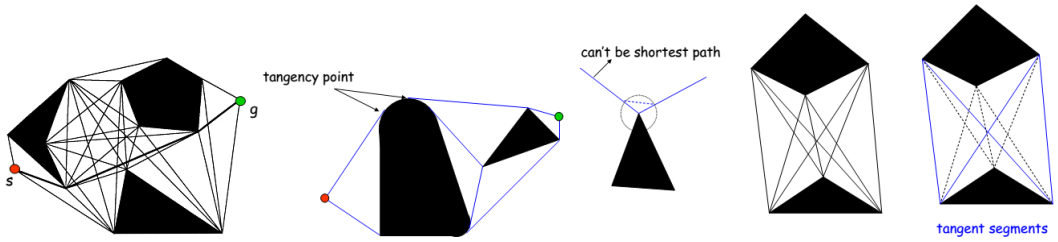
Svantaggi dei campi di forza

- annullamento delle intensità dai campi (problema di minimo loacale):problema box canyon problem
- movimenti discontinui e non uniformi

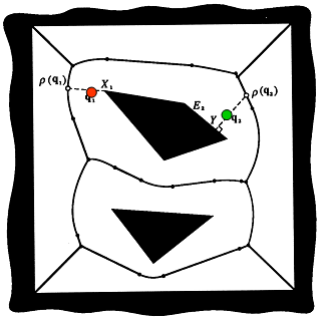
18 Probabilistic roadmaps

Una probabilistic road-map è una rappresentazione discreta di una configurazione di spazio continua generata campionando in modo casuale le configurazioni libere dello spazio e connettendo questi punti in un grafo.
La road-map si basa su diversi metodi:

- **Visibility graph**
  - E' un grafo che rappresenta l'ambiente come un insieme di poligoni.
  - Il metodo calcola il percorso più corto tra i due punti senza scontrarsi con gli ostacoli. Per farlo valuta solo i percorsi rettilinei che uniscono i vertici dei vari ostacoli e il punto iniziale e finale.
- **Voronoi diagram**
  - data una mappa in input, si generano dei punti caratteristici (e.g. punto medio tra due ostacoli nella mappa) da far seguire al robot durante il suo movimento.
  - è un algoritmo che trova un percorso la cui distanza media dagli ostacoli è massima ed è un planner completo.



(a) Visibility graph.



(b) Voronoi diagram

Figura 104: Metodi di road-mapping.

18.1 Probabilistic Road Map (PRM)

La costruzione della porbabilistic road map segue i seguenti punti

1. Si estraggono ***casualmente*** dei punti (*milestone*) nello spazio libero, fino a dire che si è ben discretizzato lo spazio:
   (a) Per ogni *milestone* si verifica con la cinematica diretta se la posizione è ammissibile oppure è da scartare perchè collide con qualche ostacolo (questa parte richiede una rappresentazione 3D dell'ambiente e dei link del robot).
   (b) si cerca inoltre di posizione delle *milestone* anche nelle regioni più «difficili» usando funzioni euristiche pesate $w(c)$.
2. Con le *milestone*, accettabili, si procede con la costruzione di un grafo.

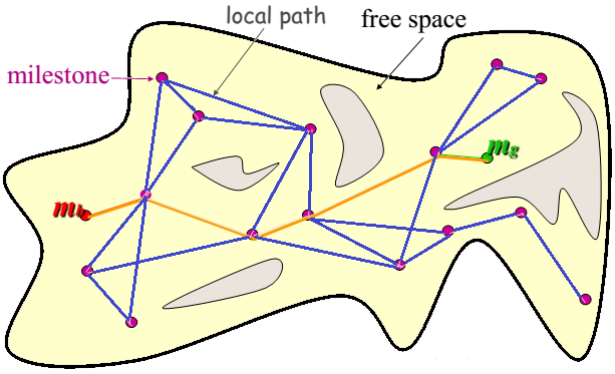


Figura 105: Proabilistic Road Map.

(a) si scelgono delle coppie di punti, casualmente, tra quelle disponibili

(b) si generano dei punti intermedi, a distanze casuali, che stanno sul segmento che connette la coppia.

(c) se i punti intermedi generati collidono con l'ostacolo oppure finiscono nello spazio degli ostacoli, si scarta l'arco di connessione della coppia, altrimenti lo si accetta e si connettono le due *milestone* (PARTE NON EFFICIENTE).

3. Una volta costruito il grafo si procede con la ricerca del percorso per raggiungere la milestone finale.

18.1.1 Learning phase

E' la fase in cui si costruisce la mappa e si divide in tre fasi:

- **Costruzione**
  In cui si prendono i punti del grafo e si crea un grafo connesso
- **Espansione**
  Si identificano le zone critiche e si prendono più punti.
- **Riduzione(facoltativa)**
  Si tolgono i collegamenti inutili e ridodanti

Construction Step

- Si parte con una mappa vuota $R = (N, E)$
- Si seleziona un punto c a caso e lo si aggiunge a N
  La scelta del nodi:
  - che siano equi distribuiti in tutto lo spazio
  - verificare che non ci siano collisioni (effettuare un collision checking).
- Si seleziona un sottoinsieme N_C di N di nodi vicino a c che si tenterà di connettere a c
  Bisogna scegliere questi nodi bene in modo da non passare al planner cose che si sa che falliranno.
  Per farlo si usano metodi euristici
  - I nodi sono più vicini di una certa soglia
  - Si evitano di prendere i nodi già connessi tra di loro perchè tanti il percorso ci sarebbe già
  - Si sceglie un numero massimo di nodi
- Si prova a unire il punto c a quelli dentro N_c usando un local planner

  - Se si usa un bel local planner la connessione avviene quasi di certo e quindi si possono anche prendere meno punti, però diventa lenta la computazione.
  - Se si usa un planner di merda (per esempio li connette in linea retta) sono necessari più punti perchè magari non riesce a connetterli, ma la computazione è molto veloce

- I collegamenti fatti si uniscono vengono aggiunti a E

Expansion step

- Si parte con una mappa $R = (N, E)$ che copre in modo equiprobabile tutto lo spazio
- Si cercano i nodi messi in una zona difficile valutando con una funzione di peso euristica $w(c)$
  Alcuni criteri euristici usati per la funzione w sono:
  - w è inversamente proporzionale al numero di nodi all'interno di una certa zona vicina a c
  - w è direttamente proporzionale al numero di fallimenti del local planner
  - w è inversamente proporzionale alla distanza del nodo connesso a c più vicino
- Si aggiungono altri nodi usando il random-bounce walks
  - Si sceglie una direzione a caso e si prosegue rimbalzando sugli ostacoli in quella direzione finchè non si incontra un nodo
  - Se quella direzione non incontra un nodo entro un certo tempo si prova con un altro nodo
  - quando si incontra un nodo si aggiungono i vertici del rimalzo e la strada fatta alla road map
- Si ripete finchè non si è contenti

18.2 Query phase

è la fase in cui si cerca nel grafo la strada un algoritmo usato è il rapidly growing random tree:

- Si costruisce un albero con radice il punto inziale e nodi presi dalla road map
- Si prende un nodo a caso dell'albero
- Si estende l'albero a partire dal nodo scelto in una determinata direzione
- Ci si ferma quando si trova l'arrivo

I pianificatori PRM funzionano bene perchè danno maggiore priorità a percorsi che passano per punti con alta visibilità (molti nodi).

Idea +
costruire degli alrberi di milestone, partendo da punti diversi dello spazio libero.
In questo modo prima o poi i due alberi si incontreranno (le foglie coincidono), e si otterranno due alberi pieni di configurazioni connessi tra loro.
Questa tecnica è molto efficacie ma ha costi altissimi. Infatti è un processo che si svolge offline.

18.3 Kinodynamic Planning

Tutto quello visto funziona se il robot può modificare a piacimenti i parametri dello spazio delle configurazioni, ma questo non è sempre vero.
Infatti esistono anche i **robot underactuated** in cui ci sono più parametri di configurazione che attuatori, tipo il differential drive.
Per questi sistemi ci sono due strategie per il planning:

- <u>Two phase planning</u>
  Si trova il percorso con i metodi visti e poi loi si approssima in modo che il robot possa seguirlo.
  - Questo sistema potrebbe comportare al robot di fare movimento strani per poter raggiungere il target, tipo parcheggio a esse.

- Direct planning
  Invece di prendere punti a caso si scelgono i punti in un albero i cui archi sono movimenti possibili e la radice è il nodo iniziale. In questo modo funziona tutto. Questo si ottiene in diversi modi:
  - Si prende un determinato vettore di controllo e si integra l'equazione del modo per una piccola durata e si prende un nuovo punto.
  - Questo sistema non garantisce di arrivare esattamente al punto e quindi si utilizza una regione di arrivo

18.4 Sampling strategies

Obbiettivo Fino a ora si è sempre preso i punti per le road map a caso, ma esistono metodi più efficienti in modo da avere una mappa con meno punti, ma altrettanto efficace.

Per farlo si utilizza una strategia di sampling non uniforme che da priorità alle zone critiche come i passaggi stetti.

18.4.1 The Gaussian Sampling Strategy

Si procede nel seguente modo:

1. Si sceglie un punto a caso c_1
2. Si prende un punto c_2 distante d dal punto c_1
3. Se c_1 e c_2 sono entrambi non raggiungibili non si fa nulla
4. Se c_1 e $c_2$sono entrambi raggiungibili non si fa nulla
5. Se c_1 è raggiungibile e c_2 no, si aggiunge c_1
6. Se c_2 è raggiungibile e c_1 no, si aggiunge c_2
7. si cicla finchè non si è contenti

Il termine d è un parametro che permette di cambiare il funzionamento dell'algoritmo:

- Con d grande si torna a prendere i campioni a caso.
- Con d piccolo si prendono campioni solo vicino agli ostacoli.

In generale questo sistema campiona tutto e tiene solo quelli vicino ai bordi.

18.4.2 The Bridge Test for Sampling Narrow Passages

Si procede nel seguente modo:

1. Si sceglie un punto a caso c_1
2. Se c_1 è un punto libero si scarta
3. Si prende un punto c_2 distante d dal punto c_1
4. Se c_2 è punto libero si scarta
5. Se c_1 e c_2 sono entrambi occupati si prende il punto medio m tra i due.
6. Se m è libero si aggiunge m
7. si cicla finchè non si è contenti

Questo sistema prende molti punti nei passaggi stretti, ma troppo pochi nel resto dello spazio e quindi solitamente lo si unisce con un sampling uniforme che prende più punti nello spazio libero.

18.4.3 Bridge Test Hybrid

Si uniscono i 3 modi visti tipo:

1. Si prendono 2 punti c_1 e c_2
2. Se sono entrambi liberi allora li si tiene con una certa probabilità (sampling uniforme)
3. Se uno dei due è occupato si tiene l'altro con una certa probabilità (sampling gaussiano)
4. Se sono entrambi occupati si mette il punto medio con una certa probabilità (bridge sampling)

19 Controllo assi

Obbiettivo Dato un percorso o una traiettoria che un braccio deve eseguire farla seguire al braccio.
Il percorso può essere comunicato tramite un insieme di punti nello spazio di configurazione o uno nello spazio cartesiano.

Siano date le seguenti definizioni

- ***percorso (path)***: una sequenza di configurazioni dei giunti di un robot in un particolare ordine.
- ***traiettoria***: definisce la tempistica con cui un percorso viene eseguito, cioè gli istanti in cui il robot deve raggiungere le varie configurazioni del percorso.

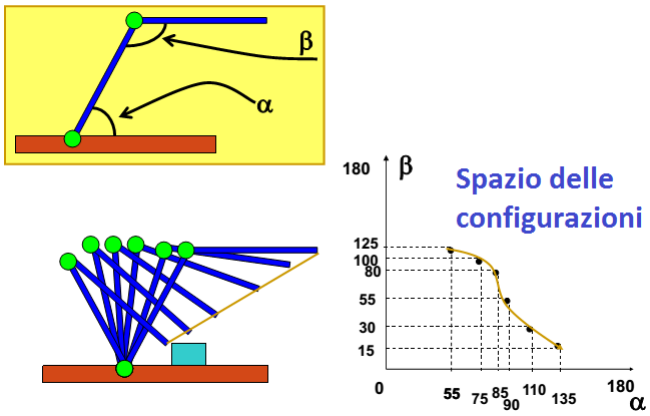


Figura 106: Spazio dei giunti e spazio cartesiano.

19.1 Spazio cartesiano

Richiede di suddividere la traiettoria del polso in tanti tratti per calcolare la cinematica inversa di ogni punto intermedio.

- ☺ è un metodo **semplice**
- ☹ **non** è **efficiente** perchè bisogna calcolare la cinematica inversa e interpolare continuamente.
- ☹ è **pericoloso** perchè è difficile garantire che una traiettoria non contenga punti singolari.

Il controllore cartesiano usa il modello cinematico. Per considerare anche la velocità è necessario la Jacobiana, così da consentire il passaggio: variabili cartesiane $\Longleftrightarrow$ variabili di giunto. Il calcolo della Jacobiana è pesante e critico. Per ovviare questo problema si ricorre al controllo nello spazio dei giunti.

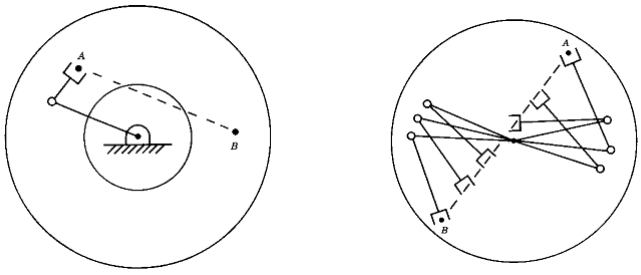


Figura 107: *Punti intermedi non raggiungibili* **e** *traiettoria cartesiana con dei punti di singolarità.*

19.2 Spazio giunti

Richiede di calcolare la cinematica inversa solo dai punti di partenza e arrivo di una traiettoria.

- ☹ è un metodo **efficiente**

- ☹ è **complesso** perchè la traiettoria del polso del robot è difficilmente correlabile alla traiettoria dei giunti.

19.2.1 Calcolo della traiettoria singolo asse

La cinematica definisce una sola configurazione per il punto B, ma in questo caso ci sono due traiettorie possibili per raggiungere B da A e una contiene anche un ostacolo. Potrebbe essere una buona idea quella di specificare un punto di passaggio così da evitare l'ostacolo.

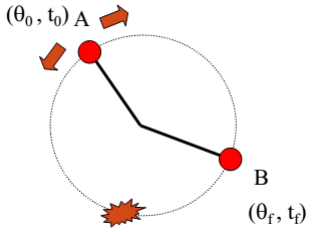


Figura 108: Robot con un solo grado di libertà.

La traiettoria $\theta(t)$ viene descritta da un polinomio di terzo grado $\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$, in cui i quattro coefficienti indicano la forma della traiettoria e si possono variare in infiniti modi.
I quattro coefficiente altro non sono che i vincoli di movimento per il giunto rotazionale in questione, e nel caso generale sono:

- A si trova in posizione θ_0 con velocità nulla: $\theta(0) = \theta_i$, $\dot{\theta}(0) = \theta_i = 0$

- B si trova in posizione θ_f con velocità nulla: $\theta(t_f) = \theta_f$, $\dot{\theta}(t_f) = \theta_f = 0$

$$
\begin{aligned}
\theta(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\
\dot{\theta}(t) &= a_1 t + 2a_2 t + 3a_3 t^2 \\
a_0 &= \theta_i \\
a_1 &= \dot{\theta}_i \\
a_2 &= \frac{3(\theta_f - \theta_i) - (\dot{\theta}_i - \dot{\theta}_f) \cdot t_f}{t_f^2} \\
a_3 &= \frac{2(\theta_i - \theta_f) - (\dot{\theta}_i + \dot{\theta}_f) \cdot t_f}{t_f^3}
\end{aligned}
$$

Punto di passaggio: fornisce un vincolo del tipo $\dot{\theta}\,(t_{via}) = \omega$.

19.2.2 Calcolo della traiettoria con una velocità costante in mezzo

Si vuole calcolare una traiettoria che abbia una velocità costante V nella parte centrale. Una funzione quadratica richiede 3 vincoli

- due per il punto di partenza

- uno per la velocità costante alla fine del tratto

Assumiamo che i due tratti parabolici siano simmetrici e che si congiungano con il tratto lineare agli istanti t_b e $t_f - t_b$.

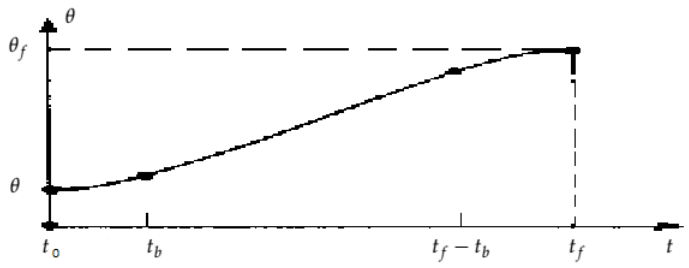


Figura 109: Traiettoria a velocità costante a tratti.

- $\theta(0) = \theta_i$: posizione iniziale;

- $\dot{\theta}(0) = 0$: velocità iniziale;

- $\dot{\theta}(t_b) = \omega$: velocità costante al termine del primo tratto parabolico;

- $\dot{\theta}(t_f - t_b) = \omega$: velocità costante all'inizio del secondo tratto parabolico;

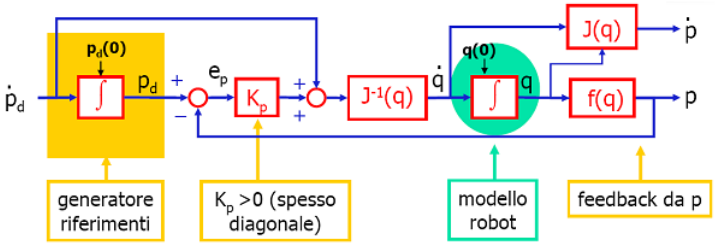
- $\theta(t_f) = \theta_f$: posizione finale;

- $\dot{\theta}(t_f) = \theta_f$: velocità finale;

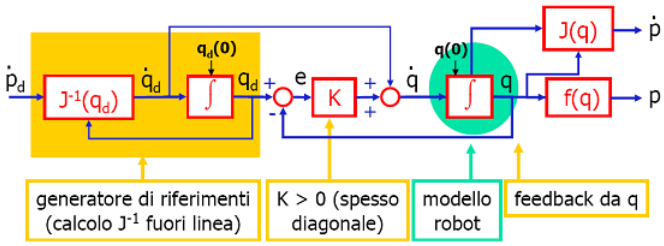
$$
\begin{aligned}
\theta(t) &= a_0 + a_1 t + \frac{1}{2} a_2 t^2 \quad \longrightarrow \quad a_0 = \theta_i \\
\dot{\theta}(t) &= a_1 + a_2 t \quad \longrightarrow \quad a_1 = 0 \\
\dot{\theta}(t_b) &= a_2 t_b = \omega \quad \longrightarrow \quad a_2 = \frac{\omega}{t_b} \\
\theta(t) &= \theta_i + \frac{1}{2} \frac{\omega}{t_b} t^2 \\
\mathbf{t_b} &= \frac{\theta_i - \theta_f + \omega \cdot t_f}{\omega}
\end{aligned}
$$

19.2.3 Controllo cinematico di un asse

Si vuole realizzare effettivamente un moto desiderato nonostante la presenza di disturbi, incertezze sui parametri del robot ed errori iniziali. Lo schema di controllo misto composto da *feedback* (misurando lo stato interno del robot) e *feedforward* (comandi nominali pianificati).
L'errore può essere definito nello spazio cartesiano o dei giunti, il comando finale è sempre a livello dei giunti.



(a) Controllo cinematico nel cartesiano.



(b) Controllo cinematico nei giunti.

Figura 110: Schemi di controllo cinematico.

Come ingresso si considera un comando cinematico, tipicamente la velocità. Ciò è possibile grazie alla presenza di un anello di controllo di basso livello, che impone idealmente qualsiasi velocità di riferimento comandata. Tale anello è già presente in una struttura chiusa di controllo, in cui l'utente può agire solo con comandi di tipo cinematico.
Le prestazioni sono in genere soddisfacenti se non si richiedono movimenti troppo veloci e/o con brusche accelerazioni.

20 Controllo in forza

Obiettivo Dato che un end effector può modificare l'ambiente toccandolo, bisogna avere dei sensori e un controllo per evitare che il braccio applichi troppa o troppo poca forza per eseguire un'azione.

Per questo motivo si parla di force control.
Viene realizzato con un controllo in retroazione che segue il movimento del robot e ne regola la velocità istante per istante.

Nomenclatura

- Tool center point (TCP) è il posto dell'end effector in cui è applicata la forza.
- remote center of compliance device è il punto dove bisogna applicare al forza.

Controllo passivo Sull'end effector è presente un dispositivo meccatronico che adatta la forza in automatico.

Controllo attivo L'end effector è dotato di un sensore di forza e nel robot è presente una legge di controllo.

21 Collision detection

Obbiettivo Avere un algoritmo che capisce che due oggetti si stanno scontrando.

Ipotesi Si ha a disposizione una point clound con i punti uniti a triangolo.

21.1 Algoritmo

Si divide in 3 parti:

- Approssima gli oggetti in sfere usando un albero binario di palle.
- Calcola la distanza minima tra due oggetti usando il branch and bound.

21.1.1 Creazione dell'albero di palle

- Si ricopre la superficie dell'oggetto con sfere piccole con un raggio predefinito.
- Si cerca la sfera più piccola possibile che contenga due delle sfere aggiunte, queste sfere possono sovrapporsi.
- Una volta che tutte le palline sono inserite dentro una palla si aggiungono all'albero queste palle che hanno come figlie le palline al loro interno.
- Si ripete fino ad avere una sola palle grande che fa da root.

21.1.2 Branch and bound

Dato l'albero di palle per calcolare la distanza da un punto si usa il branch and bound procedendno nel seguente modo:

- Si parte dalla root e si calcola la distanza minima tra la palla della root e quella del punto.
- Si cerca tra gli immediati figli quello che è più vicino al punto e si calcola tale distanza.
- Si ripete la stessa cosa partendo dal nodo scelto

Si continua fino ad arrivare alle foglie ottenendo così la distanza minima. Questo algoritmo è abbastanza efficiente infatti la sua complessità è $O(n \cdot \log n)$ in media e $O\left(n^2\right)$ nel peggiore dei casi dove n è il numero di foglie.

21.1.3 Collision Detection

Si continua iteramente controllando quale delle palle si intersecano e se si arriva a un punto in cui non si intersecano allora non c'è collisione. Mentre se si arriva alle foglie si controlla se la superficie vera dell'oggetto all'interno della sfera collide o no.

Quindi è una algoritmo veloce se non si intersecano, ma è lento se bisogna arrivare fino in fondo.