Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Cognitive SLAM: knowledge-based self-localization and mapping

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore:
Prof. Andrea Bonarini

Tesi di Laurea di:
Davide Tateo Matricola n. 799311

Anno Accademico 2013-2014

A qualcuno...

Sommario

Lo SLAM è uno dei principali problemi nello sviluppo di robot autonomi. Gli approcci correnti sono afflitti da un pesante costo computazionale e si comportano male in abienti dinamici e disordinati; le performance di questi sistemi diventano ancora peggiori se si utilizzano sensori a basso costo, necessari per le applicazioni commerciali. Questa tesi affronta il problema da un punto di vista nuovo e originale, usando feature ad alto livello come punti chiave e sfruttando la conoscenza di un esperto e un linguaggio fuzzy per riconoscerli, per tenere traccia di punti chiave forti e stabili e permettere mappe più intelligenti e una localizzazione robusta in ambienti complessi. L'idea fondamentale è quella di mantenere il tasso di errore della localizzazione limitato e ridurre il costo necessario a far navigare con successo un robot autonomo in un ambiente interno, usando solo i dati provenienti da una webcam e una unità di misura inerziale a basso costo. Il principale problema è il riconoscimento di feature ad alto livello, come porte e scaffali, affrontato tramite un classificatore fuzzy ad albero, definito da un esperto, in modo da evitare fasi di allenamento e migliorare la generalità del riconoscimento.

Abstract

SLAM is one of the key issues in autonomous robots development. Current approaches are affected by heavy computational load and misbehave in cluttered and dynamic environment; their performance get even worse with low cost sensors, needed for market applications. This thesis faces the problem in a new and original way, working with high level features as key points and using expert knowledge and a fuzzy language to detect them, in order to track strong and stable key points and allow smarter maps and robust localization in complex environments. The key idea is to keep the error rate of the localization process limited, and to reduce the cost of an autonomous robot to successfully navigate into an indoor environment using only the data from a webcam and a low cost Inertial measurement unit. The main issue is the high level feature recognition, like doors and shelves, done by an expert-defined fuzzy tree classifier, in order to avoid training and improve generalization of recognition.

Indice

Ringraziamenti

Ringrazio .....

Capitolo 1

Introduzione

"Qualunque cosa che accade, accade"
"Qualunque cosa che, accadendo, ne fa accadere un'altra, ne fa accadere un'altra."
"Qualunque cosa che, accadendo, induce se stessa a riaccadere, riaccade."
"Però non è detto che lo faccia in ordine cronologico."

Douglas Adams, Mostly Harmless

Uno dei problemi chiave della robotica è la localizzazione di un robot in un ambiente sconosciuto. Questo problema è noto come "SLAM", Simultaneus localization and Mapping, ed è attualmente una delle aree più importanti della ricerca riguardante i robot autonomi. In particolare, questo problema risulta ancora più difficile se applicato a robot dotati di sensori a basso costo, quali webcam e unità di misura inerziale economiche, restrizione fondamentale per la diffusione di applicazioni di robotica autonoma nel mondo reale.
Attualmente questo problema è affrontato attraverso tecniche probabilistiche, che sono ottime per analizzare ambienti statici e per ottenere una localizzazione precisa. Le mappe create sono utilizzabili principalmente per la navigazione. I sensori più usati sono gli scanner laser, che tuttavia sono molto costosi, i sensori RGB-D, sensori economici che nascono per applicazioni videoludiche commerciali e hanno alcune limitazioni se applicati alla robotica, le telecamere stereo.
Lo scopo della tesi è di sviluppare un framework per risolvere il problema della localizzazione di robot autonomi dotati di sensori a basso costo, quali ad esempio i quadricotteri. L'idea è di sviluppare una metodologia che non solo permetta al robot di localizzarsi nella mappa in maniera efficiente, ma anche di interagire con l'ambiente in maniera avanzata, di compiere ragionamenti, di adattarsi a eventuali cambiamenti ed agli eventi che possono

occorrere, quali la presenza di persone o altri agenti.
Basandosi principalmente sulle informazioni provenienti da una videocamera monoculare, è stato sviluppato un sistema che è in grado di processare informazioni provenienti da un esperto, espresse in un linguaggio formale, di estrarre feature di basso livello e aggregarle, grazie alla base di conoscenza, per riconoscere e tracciare feature di alto livello, quali porte e armadi, e di basare conseguentemente su di essi il processo di localizzazione. Per permettere all'esperto di trasmettere la sua conoscenza all'agente, è stato sviluppato un linguaggio formale, basato sulla logica fuzzy, che permetta di esprimere sia regole fuzzy linguistiche, sia di definire un classificatore fuzzy ad albero in grado di definire una gerarchia di modelli e le relazioni tra di essi.

La tesi è strutturata nel modo seguente:
Nel capitolo 2 si illustra lo stato dell'arte.
Nel capitolo 3 si illustrano le basi teoriche necessarie.
Nel capitolo 4 si descrive l'architettura generale del sistema.
Nel capitolo 5 si parla del reasoner e del linguaggio formale utilizzato.
Nel capitolo 6 si descrive l'estrazione delle feature di basso livello.
Nel capitolo 7 si mostra come gli oggetti riconosciuti sono tracciati.
Nel capitolo 8 si illustra la creazione della mappa e la localizzazione.
Nel capitolo 9 si analizzano i risultati sperimentali del sistema proposto.
Nel capitolo 10 si riassumono gli scopi, le valutazioni di questi e le prospettive future.

Capitolo 2

Stato dell'arte

Doc: Ecco perché non ha funzionato: c'è scritto "Made in Japan".
Marty: E che vuol dire Doc? Tutta la roba migliore è fatta in Giappone.
Doc: Incredibile!

Ritorno al Futuro, parte III

2.1 SLAM

Il problema della localizzazione di un robot e costruzione della mappa in contemporanea in un ambiente sconosciuto è stato affrontato fin dagli anni '90 a partire da [23], articolo nel quale per la prima volta si delineava un framework per localizzare un robot costruendo contemporaneamente la mappa dell'ambiente.

Tramite l'utilizzo di sonar, venivano estratte feature geometriche con cui veniva costruita la mappa, nella quale il robot si localizzava. Il problema principale della localizzazione è il "problema della correlazione": se la posizione della feature rispetto alla quale ci si localizza è affetta da incertezza, la conseguente stima della posizione effettuata rispetto a tale feature sarà affetta da un errore che dipende dall'errore della posizione della feature stessa. Questo problema diventa tanto più grave se si pensa che la posizione del robot in ogni istante non è nota a priori, ma deve essere stimata sulla base delle osservazioni precedenti.

È necessario risolvere questo problema per evitare che l'errore della generazione della mappa e l'errore della stima della posizione divergano nel tempo. Per risolverlo, gli autori hanno spesso utilizzato un filtro di Kalman esteso.

Il filtro di Kalman è uno stimatore Bayesiano ricorsivo, che, supposto noto il modello lineare che regola la generazione dei dati e la loro osservazione, supposto che l'errore di misura e di modello siano gaussiani, restituisce la densità di probabilità del sistema osservato. Il filtro di Kalman, se utilizzato secondo le ipotesi, è uno stimatore ottimo dello stato del sistema osservato, secondo i minimi quadrati. Tuttavia, nell'ambito della robotica, e in particolare nel problema della localizzazione, il modello di generazione e osservazione dei dati non può essere considerato lineare. E' quindi necessario utilizzare un'estensione del filtro di Kalman al caso non lineare: il filtro di Kalman esteso (EKF) è una delle possibili soluzioni al problema. L'idea alla base del filtro di Kalman esteso è quella di lavorare sul modello linearizzato, stimato ricorsivamente dal modello non lineare sulla base della stima corrente.

Per avere una buona stima della posizione è necessario utilizzare un gran numero di feature, numero che cresce molto rapidamente con l'aumentare della dimensione dell'ambiente. La complessità computazionale dell'approccio tradizionale basato sul filtro di Kalman esteso è $\mathcal{O}(N^3)$, con N numero di feature, e quindi il tempo di calcolo diventa ben presto inaccettabile per prestazioni in tempo reale. Per risolvere questo problema è stato introdotto in [30] un nuovo algoritmo detto FastSLAM, che consiste nell'utilizzo del Particle Filter, e del filtro di Kalman esteso in combinazione. L'algoritmo associa ad ogni feature considerata, un filtro di Kalman esteso; la densità di probabilità congiunta, invece, viene calcolata sfruttando il Particle filter. Il Particle Filter è un altro stimatore Bayesiano ricorsivo, che, invece di un modello e dell'assunzione di rumore gaussiano, sfrutta metodi di tipo Monte Carlo per stimare la densità di probabilità del sistema che genera i dati. Il risultato è un algoritmo che ha complessità computazionale $\mathcal{O}(N \log M)$, con M numero di feature e N il numero di particelle usate dal Particle Filter. Questo approccio rende il problema trattabile nella maggior parte dei casi, pur essendo pesante computazionalmente, dato che è necessario un elevato numero di particelle per avere una buona localizzazione.

Vista la particolarità del problema quando il sensore utilizzato è una videocamera monoculare, sono stati sviluppati algoritmi ad hoc. Uno degli algoritmi più usati è PTAM [19], [20]. L'idea alla base di questo algoritmo è dividere in due thread separati il tracking e la creazione della mappa: un thread si occupa del tracking robusto di feature a basso livello, mentre l'altro thread si occupa della creazione della mappa. Per rendere efficiente il processo di mapping, solo i keyframe, ossia i frame che contengono maggiore informazione rispetto a quella già presente, vengono considerati. Per rendere il processo di mapping robusto, vengono utilizzate tecniche batch per

costruire la mappa, come ad esempio il bundle adjustment. Il bundle adjustment consiste in un processo iterativo di raffinamento della stima dei punti 3D ricostruiti e della posa della videocamera. PTAM, tuttavia, nasce per applicazioni di realtà aumentata, e quindi necessita di una inizializzazione, per risolvere i problemi dell'acquisizione del primo keyframe e per gestire la scala della mappa.

Un approccio alternativo consiste nell'usare tutti i dati dell'immagine per eseguire la localizzazione, questo approccio è alla base, ad esempio, di DTAM, Dense Tracking and Mapping [32]. Questo algoritmo crea un modello denso dell'ambiente e usa l'allineamento della videocamera

Si sono dimostrati efficaci anche i metodi semi-diretti, come SVO [10], Semi Direct Visual Odometry, un algoritmo che riesce a ottenere altissime prestazioni limitando l'estrazione delle feature ad alto livello ai soli keyframe, operando direttamente sulle intensità dei pixel nei frame successivi, eliminando le fasi computazionalmente più onerose, che sono l'estrazione e l'abbinamento delle feature. SVO si basa sulle idee di PTAM, ma ne migliora sia le prestazioni, riuscendo a essere computazionalmente più leggero, sia la precisione della mappa e della localizzazione riducendo di molto gli outlier.

Recentemente, stanno avendo molto successo i sistemi basati su sensori RGB-D [16], [11]. Questi sensori vengono utilizzati come scanner laser a basso costo per creare una mappa dell'ambiente. Tuttavia, sono soggetti a molte limitazioni, non essendo stati progettati per questo scopo, e soffrono tra l'altro di un raggio d'azione limitato. Nonostante queste limitazioni i sistemi riescono comunque a ottenere buone prestazioni in ambienti indoor [39].

2.2 Rappresentazione del Mondo

Sono noti diversi modi per rappresentare un ambiente tridimensionale. Il più semplice possibile è quello di usare delle nuvole di punti, direttamente estratte dai sensori. Un'altra rappresentazione comune è quella di filtrare le nuvole di punti ottenute tramite una griglia di voxel, come, ad esempio, in [38]. Metodi più avanzati permettono una rappresentazione geometrica dell'ambiente con un minor uso di memoria, come, ad esempio, le mappe di quota, in cui una mappa a due dimensioni è estesa con il calcolo del valore medio di altezza di ciascun punto 2D [12], le mappe di quota estese [35], che tengono conto di possibili aperture attraversabili dai robot, oppure le mappe di quota multi livello [41], che riescono a descrivere complesse geometrie multi livello.

Tra le mappe più promettenti esistono le mappe basate sugli octree, che permettono un'efficiente rappresentazione in memoria sia dello spazio occupato sia dello spazio libero, pur potendo descrivere geometrie complesse. Inoltre, questo tipo di rappresentazione permette di scalare facilmente la risoluzione della mappa, permettendo di utilizzare la stessa mappa per compiti che richiedono una precisione differente. Un'efficiente implementazione di questo tipo di mappa può essere trovata in [15].

Recentemente, si sta sviluppando l'idea di rappresentare il mondo ad alto livello, incorporando informazioni semantiche e geometriche che possano essere utilizzate non solo dagli algoritmi di navigazione, ma anche per svolgere compiti ad alto livello e ragionamenti. Una possibile soluzione al problema è l'approccio basato su Scene Graph [2]. Sono stati sviluppati anche linguaggi specifici di dominio (DSL) per poter interagire ad alto livello, ad esempio generando istanze di oggetti a partire da un modello generico, come ad esempio in [3].

2.3 Riconoscimento di Oggetti

La quasi totalità degli algoritmi di riconoscimento di oggetti nell'immagine è basata su tecniche di machine learning. Una delle classi di algoritmi più usati sono quelli basati sulle Haar-like feature [34]. Queste feature sono calcolate in aree rettangolari, all'interno delle quali vengono sommati i valori dell'intensità dei pixel. Le somme sono in seguito usate per calcolare differenze tra aree di interesse, per poter riconoscere feature geometriche quali angoli, linee o bordi. Un efficiente algoritmo per il riconoscimento di oggetti è descritto in [42] ed è stato ampliato in [25], e consiste nell'utilizzare in cascata una serie di classificatori via via più restrittivi; ognuno dei classificatori di ogni stadio è costruito attraverso una tecnica di boosting, ossia sono formati da un insieme di classificatori deboli che creano un classificatore complessivo più restrittivo. I classificatori di base sono solitamente alberi di decisione. Il classificatore complessivo dà una risposta binaria; per riconoscere effettivamente l'oggetto deve essere applicato tramite una finestra mobile su tutta l'immagine.
Un algoritmo simile è descritto in [1], dove, però, invece che feature Haar-like, vengono utilizzati direttamente un insieme di pixel selezionati nell'immagine.

Un'altra importante classe di algoritmi è basata sulle feature HOG (Histogram of Oriented Gradients) [6].

Queste feature sono ricavate contando le occorrenze dell'orientamento dei gradienti in sotto-blocchi dell'immagine. Su queste feature si basa l'al-

goritmo descritto in [9]. Questo algoritmo è in grado di definire un oggetto a partire dalle sue parti, e quindi è robusto anche a occlusioni parziali dell'oggetto. Tuttavia è computazionalmente molto più pesante dell'approccio basato su feature Haar-like.

Più recenti sono i metodi basati sul deep learning. Tra questi, i più promettenti nel riconoscimento di oggetti sono basati sulle reti neurali convoluzionali. La particolarità di queste reti è di essere basata su kernel di nodi collegati in maniera fissa, e ripetuti per coprire tutta l'immagine. La struttura rigida sfrutta la località dell'informazione nell'immagine, e permette un training più efficiente. Un esempio di applicazione si può trovare in [21].

Metodi recenti, sfruttano sensori RGB-D, come ad esempio Kinect, per riconoscere oggetti nella scena, come ad esempio in [22]. La maggior parte delle tecniche usate sono un'estensione delle tecniche già descritte, spesso estendendo i descrittori delle feature grazie alle informazioni sulla profondità.

2.4 Reasoning

I sistemi di ragionamento sono stati alla base dell'Intelligenza Artificiale fin dalla nascita ed hanno trovato sbocco in diverse applicazioni tra cui i sistemi esperti, per lungo tempo una delle branche dell'intelligenza artificiale più attive [4] [7] [17].

Con la maturazione della tecnologia per lo sviluppo dei sistemi esperti, e l'avvento di Internet, l'attenzione della ricerca collegata ai sistemi di ragionamento formale si è spostata dai sistemi esperti al web semantico. Il linguaggio più diffuso per i sistemi di inferenza è il linguaggio OWL 2 [33], [13], fortemente basato sulle logiche descrittive che permettono un buon compromesso tra espressività, e complessità computazionale e mantengono la logica utilizzata decidibile.

Fin dai primi sistemi esperti si è riconosciuta l'importanza del trattamento dell'incertezza in sistemi di ragionamento che avessero a che fare con il mondo reale [7], ed in particolare si è avuta una larga diffusione in moltissimi settori della logica fuzzy [45], adottata anche nel nostro lavoro.

Un esempio di reasoning applicato al riconoscimento delle immagini può essere trovato in [27], in cui feature a basso livello vengono estratte da un meccanismo di machine learning, e utilizzate da un sistema di inferenza che sfrutta una ontologia per l'analisi ad alto livello dell'immagine. Un altro esempio di utilizzo di sistemi di inferenza nel riconoscimento di oggetti si può trovare in [14], dove viene utilizzato il linguaggio OWL per specificare

una base di conoscenza e riconoscere tramite la conoscenza di dominio gli oggetti nell'ambiente.

Capitolo 3

Concetti Fondamentali

Capitolo 4

Architettura del Sistema

Doc: Questo rende possibile viaggiare nel tempo... il flusso canalizzatore.
Marty: Flusso canalizzatore?!
Doc: Mi ci sono voluti quasi 30 anni e tutto il mio patrimonio per realizzare la visione di quel giorno... mio Dio quanto tempo è passato!
...
Marty: Questa... questa è proprio forte Doc! E' grande! Ma funziona con la benzina normale?
Doc: Sfortunamente no! Ha bisogno di un qualcosa di un pò più vivace: Plutonio.

Ritorno al Futuro, parte I

4.1 Introduzione

In questo capitolo descriveremo l'architettura del sistema implementato. Il sistema è stato progettato in maniera modulare per una serie di motivi: questo tipo di architettura permette di riutilizzare i moduli sviluppati, di estendere semplicemente il sistema con altri moduli, di sostituire eventualmente un intero modulo con un altro che compia le stesse funzioni, di comunicare in maniera semplice con altri sistemi.

Per raggiungere questo scopo, si è deciso quindi di utilizzare il middleware ROS (Robot Operating System) [36]. ROS è un middleware che, oltre ad offrire molti strumenti utili per lo sviluppo di complesse applicazioni robotiche, implementa due pattern architetturali importanti, entrambi usati nel nostro lavoro: il pattern publish-subscribe e il pattern client-server. Questi due pattern sono implementati rispettivamente tramite messaggi e servizi: i messaggi definiscono un formato comune per la pubblicazione di

dati in topic, i servizi invece specificano un'interfaccia per le chiamate a procedure remote, dichiarando gli input e gli output tra client e server. Ogni processo che viene eseguito in ROS è chiamato nodo. Per eseguire qualsiasi sistema basato su ROS, devono essere presenti tre ulteriori nodi: il nodo Master, che si occupa di garantire la comunicazione tra gli altri nodi, tramite i due pattern supportati, il server dei parametri, che implementa un dizionario condiviso accessibile tramite la rete, in cui i nodi possono memorizzare e recuperare parametri usati dai loro algoritmi a runtime, il nodo rosout, che si occupa di mantenere i log prodotti dalle applicazioni.

Il sistema sviluppato è pensato per interagire con qualsiasi tipo di robot che sia provvisto di una videocamera monoculare e una unità di misura inerziale (IMU) tramite le due interfacce standard di ROS. Esse consistono in tre topic: "imu", per i dati provenienti dall'unità di misura inerziale, "image_raw", per l'immagine proveniente dalla videocamera, "camera_info", che contiene i parametri intrinseci della videocamera, tra cui la matrice di calibrazione della videocamera utilizzata, necessaria per il funzionamento del sistema. I dati estratti dall'unità di misura inerziale devono poter essere riferiti al sistema di coordinate della videocamera, infatti l'algoritmo di visione implementato utilizza i dati provenienti dalla IMU per stimare approssimativamente la posa della telecamera. Per rendere l'algoritmo portabile, si utilizza a libreria tf di ROS. La libreria tf gestisce le trasformazioni da sistema di coordinate all'altro, pubblicando un albero di trasformazioni nel topic "tf". Negli header dei messaggi standard di ROS è definito il campo "frame_id", che descrive il nome del sistema di coordinate rispetto al quale è riferito il contenuto del messaggio. Grazie a questo, è possibile riferire i dati della IMU rispetto alle coordinate dell'immagine, purché sia nota la trasformazione tra i due frame.

4.2 Diagramma del sistema

L'architettura del sistema implementato è descritta in Figura 4.1.

Il diagramma rappresenta i nodi del sistema, segue una breve descrizione di ogni elemento per specificare la loro funzione:

image_proc si occupa di eliminare la distorsione radiale della videocamera causata dalla curvatura della lente.

c_vision_detector si occupa di estrarre possibili feature dall'immagine.

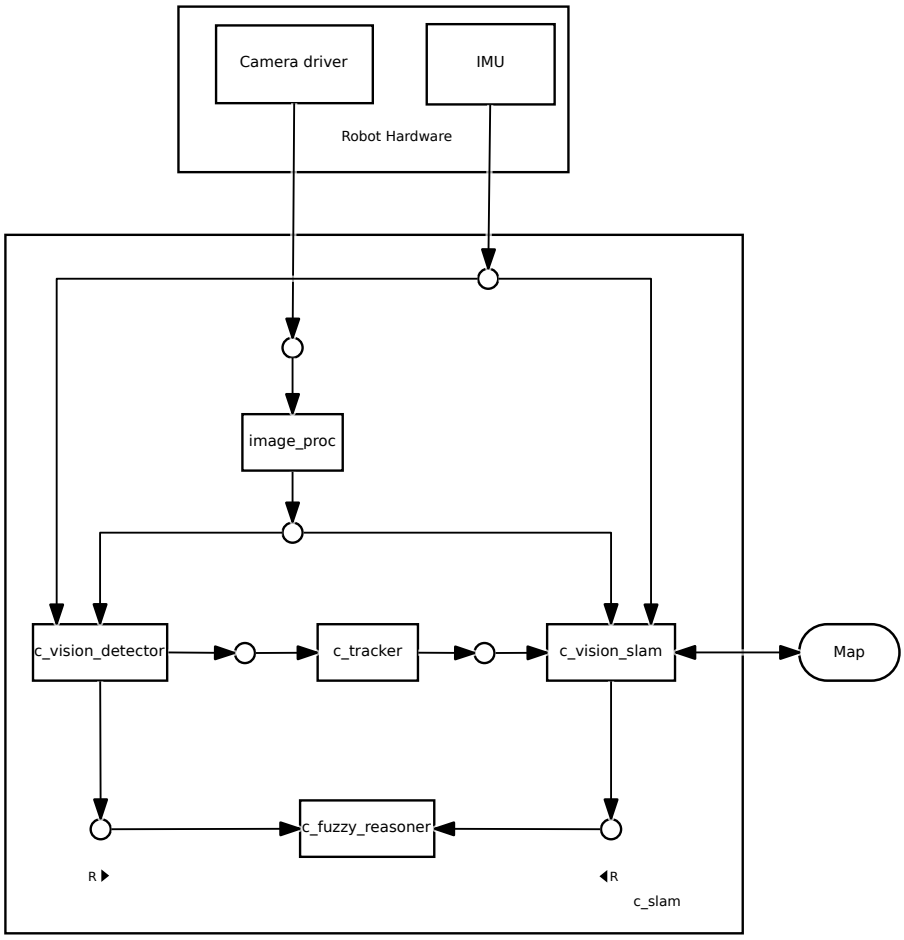

Figura 4.1: Architettura del sistema implementato

c_tracking si occupa di seguire le feature a basso livello estratte dal nodo "c_vision_detector" nell'immagine, mantenendo un modello in modo da riuscire a riconoscere la feature anche dopo essere stata persa.

c_vision_slam si occupa dell'analisi approfondita delle feature, in modo da determinare il tipo di oggetto e la sua posizione nello spazio.

c_fuzzy_reasoner implementa un reasoner fuzzy; data una base di conoscenza e un classificatore, analizza le feature in ingresso e le classifica.

4.3 Comunicazione tra i nodi

Questo sistema sfrutta sia il paradigma client-server, per l'interazione con il reasoner, che il paradigma publish-subscribe, in tutti gli altri casi. Il reasoner offre 4 servizi:

reasoning questo servizio è un normale servizio di reasoning basato su una knowledgebase fuzzy.

classification questo servizio si occupa di classificare istanze di feature riconosciute.

getDependencyGraph questo servizio restituisce il grafo delle dipendenze del classificatore.

getReasoningGraph questo servizio restituisce il grafo di reasoning utilizzato dal classificatore.

Il reasoner e i suoi servizi verranno discussi approfonditamente nel Capitolo 5.

le informazioni riguardanti gli oggetti riconosciuti e seguiti dal sistema sono scambiate tramite i seguenti topic:

to_track in questo topic vengono pubblicate le possibili feature riconosciute dall'intera immagine. Viene fornito il contorno della feature e la sua possibile classificazione.

tracks in questo topic vengono pubblicati i risultati dell'algoritmo di tracking: contiene il bounding box e il contorno della feature, quest'ultimo viene maggiorato del 20%, per assicurarsi di mantenere all'interno di esso il reale contorno della feature.

Il riconoscimento di feature viene descritto nel Capitolo 6, il tracking delle feature riconosciute nel Capitolo 7, e infine l'analisi ad alto livello viene effettuata nel Capitolo 8.

4.4 Parametri degli algoritmi

Gli algoritmi utilizzati nel sistema, in particolare gli algoritmi di visione, hanno alcuni parametri che è possibile tarare per adattare il sistema a qualsiasi tipo di robot utilizzato. Per gestire i parametri abbiamo utilizzato il server dei parametri di ROS. Il server dei parametri è un dizionario, ossia

ogni parametro, identificato da una stringa, è salvato nel server dei parametri, e può essere recuperato tramite il suo nome. Il server dei parametri supporta dizionari gerarchici, in modo da poter rappresentare tipi di dati strutturati. Inoltre è possibile definire parametri privati per qualsiasi nodo. I parametri privati restano accessibili da tutto il resto del sistema, ma nel dizionario vengono salvati sotto il nome del nodo a cui appartengono; questo comportamento è studiato per evitare collisioni tra parametri con lo stesso nome in nodi differenti.

Visto che gli algoritmi di estrazione e analisi delle feature si basano fortemente sugli stessi strumenti, questo sistema usa in maniera estesa i parametri privati, in modo da avere lo stesso parametro, che rappresenta concettualmente la stessa quantità, differente in base al nodo che lo utilizza.

Il sistema permette anche di cambiare i parametri a runtime, essi vengono aggiornati nei nodi che ne fanno uso con una frequenza fissa. Si possono conoscere i parametri usati dal sistema e il loro significato nell'Appendice A.

Capitolo 5

Reasoning

Marty: Aspetta un momento Doc. Se vado dritto verso lo schermo, andrò a sbattere contro quegli indiani!
Doc: Marty, non stai pensando quadridimensionalmente!

Ritorno al Futuro, parte III

5.1 Introduzione

In questo capitolo esporremo il funzionamento del reasoner fuzzy. Il reasoner fuzzzy si basa sulle regole fuzzy descritte da Mamdami [28]. Su questa base abbiamo implementato un classificatore fuzzy ad albero, che, data una struttura ad albero di classi di oggetti e eventuali relazioni tra di loro, accetta in ingresso delle feature, di cui sono note le caratteristiche, e le classifica. Per raggiungere questo scopo abbiamo implementato due linguaggi: uno per esprimere regole fuzzy, che permetta di esprimere le proprietà rispetto a classi di oggetti; l'altro per esprimere la struttura del classificatore, in grado di esprimere gerarchie di classi e relazioni tra esse. Inoltre abbiamo implementato un algoritmo di reasoning che permette di classificare un insieme di feature, restituendo non solo le classi a cui ciascuna feature appartiene, ma anche il grado di appartenenza di ciascuna feature alle stesse. L'algoritmo in particolare è studiato per risolvere dipendenze cicliche tra le classi e riuscire a classificare oggetti che dipendono mutuamente l'uno dall'altro.

5.2 Linguaggio Fuzzy

Il linguaggio per esprimere regole fuzzy sviluppato permette di esprimere non solo domini semplici e insiemi fuzzy su questi domini, ma anche domini

complessi, chiamati classi e predicati, validi per qualunque dominio. È ispirato fortemente al linguaggio FCL, Fuzzy Control Language, da cui prende parte della sintassi.

Tutte le variabili utilizzate dal reasoner sono assunte come variabili intere con segno, possono quindi avere valori compresi tra `INT_MIN` e `INT_MAX`. Si assume che i numeri decimali siano rappresentati semplicemente come numeri a virgola fissa.

5.2.1 Classi e Variabili

Una classe si definisce tramite la keyword `FUZZIFY_CLASS`, seguita dal nome della classe, e la sua definizione termina con la keyword `END_FUZZIFY_CLASS`. I fuzzy set riguardanti una variabile di input o output si definiscono tramite la keyword `FUZZIFY`, seguita dal nome della variabile a cui si riferiscono, e terminano con la keyword `END_FUZZIFY`. Sia le variabili di input che quelle di output possono essere definite all'interno di una classe, così facendo apparterranno alla classe in cui sono definite. Per definire un fuzzy set, si utilizza una etichetta per definire il nome del fuzzy set, seguito dal token ":=" e da una etichetta per indicare la sua forma. Ciascuna definizione è terminata dal token ";". Le possibili etichette sono:

tol "Triangle open left", fuzzy set triangolare aperto a sinistra. Necessita di due parametri.

tor "Triangle open right", fuzzy set triangolare aperto a destra. Necessita di due parametri.

tri "Triangle", fuzzy set triangolare. Necessita di tre parametri.

tra "Trapezoid", fuzzy set trapezoidale. Necessita di quattro parametri.

int "Interval", intervallo. Necessita di due parametri.

sgt "Singleton", singolo valore. Necessita di un parametro.

I parametri necessari si indicano tra parentesi, separati da virgole. I nomi delle classi e dei fuzzy set devono cominciare con la lettera maiuscola, mentre i nomi delle variabili possono essere anche minuscoli.

Ad esempio per definire sulla variabile "Input" i fuzzy set "Near", "Medium" e "Far" si scrive:

```
FUZZIFY Input
 Near := tol(100, 200);
```

```
 Medium := tra(100, 200, 300, 400);
 Far := tor(300, 400);
END_FUZZIFY
```

Mentre per definire una classe chiamata “TestClass”, con due membri “a” e “b”, si può procedere come segue:

```
FUZZIFY_CLASS TestClass
 FUZZIFY a
  Set1 := tol(100, 200);
  Set2 := tra(100, 200, 300, 400);
  Set3 := tor(300, 400);
 END_FUZZIFY

 FUZZIFY b
   Fuzzy1 := tol(100, 200);
   Fuzzy2 := tri(100, 200, 300);
   Fuzzy3 := tor(200, 300);
 END_FUZZIFY
END_FUZZIFY_CLASS
```

5.2.2 Regole Fuzzy

Le regole di Mamdami sono formate da un antecedente e da un conseguente. Nel nostro linguaggio si definiscono tramite le due keyword `IF` e `THEN`. Dopo il token `IF` ci deve essere una formula logica ben formata, dopo il secondo deve esserci un assegnamento a una variabile.

Una formula ben formata è definita come segue:

1. Un assegnamento è una formula ben formata
2. Un predicato è una formula ben formata
3. se A è una formula ben formata, anche “not A”, “(A)” sono formule ben formate
4. se A e B sono formule ben formate, anche “A and B”, “A or B” sono formule ben formate.
5. null’altro è una formula ben formata.

Un assegnamento di una variabile è composto dal nome della variabile, seguita dalla keyword `IS`, seguita da un fuzzy set da assegnare alla variabile,

tutto racchiuso tra parentesi tonde. Per utilizzare una variabile di una classe si utilizza la notazione "." tipica di molti linguaggi di programmazione orientati agli oggetti.

Un esempio di regola fuzzy è la seguente:

```
if (Input1 is Low) and (Input2 is Medium) then (Output is High);
```

5.2.3 Predicati

Si possono anche definire predicati unari. Un predicato unario si definisce tramite la keyword `FUZZIFY_PREDICATE`, seguita dal nome della variabile che si userà nel predicato (che deve necessariamente cominciare con un punto interrogativo). La definizione di un predicato unario termina con la keyword `END_FUZZIFY_PREDICATE`. All'interno di un predicato si devono definire i fuzzy set della variabile usata dal predicato. Più predicati possono essere definiti sulla stessa variabile di input, per farlo basta definire ciascun predicato con un un nome (che cominci con la lettera maiuscola), seguito dal token ":=", seguito da una formula logica ben formata, che può usare la variabile di predicato definita, terminata dal token ";".

Ad esempio per definire due predicati, "Predicate1" e "Predicate2", si può procedere come segue:

```
FUZZIFY_PREDICATE ?x
 Predicate1 := (?x is Set1) or (?x is Set2);
 Predicate2 := (?x is Set1) and (?x is Set2);

 FUZZIFY ?x
   Set1 := tol(0, 100);
   Set2 := tor(0, 100);
 END_FUZZIFY

END_FUZZIFY_PREDICATE
```

I predicati si chiamano usando il loro nome e aggiungendo tra parentesi la variabile sulla quale si vuole valutare il predicato. ad esempio se si vuole valutare il predicato "Predicate1" sulla variabile "a" si scriverà:

```
Predicate1(a)
```

5.2.4 Grammatica

⟨fuzzyFile⟩ ⊨ ⟨fuzzyDefinitions⟩ ⟨ruleSet⟩

⟨fuzzyDefinitions⟩ ⊨ ⟨fuzzyClass ⟩⟨fuzzyDefinitions⟩ | ⟨fuzzySet⟩ ⟨fuzzyDefinitions⟩ | ⟨fuzzyPredicate⟩ ⟨fuzzyDefinitions⟩ | ϵ

⟨fuzzyClass⟩ ⊨ FUZZIFY_CLASS *ID* ⟨fuzzyClassDefinitions⟩ END_FUZZIFY_CLASS

⟨fuzzyClassDefinitions⟩ ⊨ ⟨fuzzySet⟩ ⟨fuzzyClassDefinitions⟩ | ⟨fuzzyPredicate⟩ ⟨fuzzyClassDefinitions⟩ | ϵ

⟨fuzzyPredicate⟩ ⊨ FUZZIFY_PREDICATE ⟨templateVar⟩ ⟨fuzzyPredicateList⟩ ⟨fuzzyTemplateSet⟩ END_FUZZIFY_PREDICATE

⟨fuzzyPredicateList⟩ ⊨ ⟨fuzzyPredicateDef⟩ ⟨fuzzyPredicateList⟩ | ⟨fuzzyPredicateDef⟩

⟨fuzzyPredicateDef⟩ ⊨ *ID* := ⟨wellFormedFormula⟩ ;

⟨fuzzyTemplateSet⟩ ⊨ FUZZIFY ⟨templateVar⟩ ⟨fuzzyTerm⟩ END_FUZZIFY

⟨fuzzySet⟩ ⊨ FUZZIFY ⟨fuzzyId⟩ ⟨fuzzyTerm⟩ END_FUZZIFY

⟨fuzzyId⟩ ⊨ ⟨var⟩ | ⟨var⟩ , ⟨fuzzyId⟩

⟨fuzzyTerm⟩ ⊨ *ID* := *F_LABEL* ⟨shape⟩ ; | *ID* := *F_LABEL* ⟨shape⟩ ; ⟨fuzzyTerm⟩

⟨shape⟩ ⊨ (⟨parametersList⟩)

⟨parametersList⟩ ⊨ *INTEGER* | *INTEGER* , ⟨parametersList⟩

⟨ruleSet⟩ ⊨ ⟨rule⟩ ⟨ruleSet⟩ | ϵ

⟨rule⟩ ⊨ IF ⟨wellFormedFormula⟩ ⟨fuzzyAssignment⟩ ;

⟨wellFormedFormula⟩ ⊨ ⟨fuzzyComparison⟩
| ⟨fuzzyPredicateCall⟩
| (⟨wellFormedFormula⟩)
| not ⟨wellFormedFormula⟩
| ⟨wellFormedFormula⟩ or ⟨wellFormedFormula⟩
| ⟨wellFormedFormula⟩ and ⟨wellFormedFormula⟩

⟨fuzzyComparison⟩ ⊨ (⟨variable⟩ IS *ID*) | (⟨templateVar⟩ IS *ID*)

⟨fuzzyPredicateCall⟩ ⊨ *ID* . *ID* (⟨variable⟩) | *ID* (⟨variable⟩)

⟨fuzzyAssignment⟩ ⊨ THEN (⟨variable⟩ IS *ID*)

⟨variable⟩ ⊨ *ID* . ⟨var⟩ | ⟨var⟩

⟨var⟩ ⊨ *ID* | *VAR_ID*

⟨templateVar⟩ ⊨ ? ⟨var⟩

5.3 Reasoning

Il reasoning avviene utilizzando le regole fuzzy descritte da Mamdami. Il processo può essere riassunto nel seguente modo:

1. Vengono considerate tutte le variabili di input attive, ossia presenti in ingresso.

2. vengono attivate, nell'ordine in cui sono specificate, le regole che contengono solo le variabili attive.

3. viene calcolato il valore di verità del lato sinistro di ogni regola. Per farlo si calcolano i valori di verità dei fuzzy set associati alle variabili e poi si utilizzano gli operatori complemento, T-norm, S-norm per calcolare il valore di verità dell'espressione, utilizzando la normale precedenza degli operatori (not, and or).

4. Il valore di verità del lato destro della regola viene utilizzato come valore di verità per il fuzzy set della variabile di uscita.

5. I valori di verità delle variabili di uscita rispetto a un determinato fuzzy set vengono aggregati.

6. Viene defuzzificato il valore di ciascuna variabile di uscita.

Come T-norm viene utilizzato l'operatore min. Questa scelta è stata fatta perché le altre alternative conosciute, tra cui la più nota è il prodotto tra i valori di verità dei due operandi, rendono molto facilmente irrilevante il valore di verità risultante, se anche solo uno dei due operandi ha un basso valore di verità. Questo inconveniente diventa più rilevante in caso di regole lunghe, dove il valore di verità calcolato degrada di operazione in operazione. Il maggior inconveniente di questo approccio è che il valore di verità risultante dipenderà, di volta in volta, da un unico operando anziché da entrambi. Per le stesse ragioni, è stato preferito l'operatore max alla somma probabilistica o ad altre alternative.

L'operatore di aggregazione è implementato utilizzando la media semplice.

La defuzzificazione è implementata tramite l'operatore centro di massa. Per rendere più semplice ed efficiente la defuzzificazione, il reasoner si aspetta che i fuzzy set assegnati alle variabili in output siano singleton. Il vantaggio di utilizzare singleton rispetto a un generico fuzzy set è che è più semplice capire qual'è il valore massimo e minimo di una determinata variabile in uscita, infatti la relazione che lega il valore di verità di ciascuna variabile linguistica rispetto ai fuzzy set considerati con il valore calcolato in uscita è lineare. Questo in generale permette un efficiente design della knowledgebase, in quanto non si devono tenere in conto gli effetti delle non linearità delle uscite e non è quindi necessario attribuire un peso differente alle diverse regole.

Il reasoner implementato non fa differenza tra variabili di ingresso e variabili di uscita. L'unico requisito per le variabili di uscita è di assegnare come etichetta a una variabile che compare nel lato destro della regola un singleton. E' quindi possibile definire per ciascuna variabile due insiemi di fuzzy set, uno per permettere di assegnare fuzzy set alle variabili linguistiche dato un valore nel dominio, l'altro per poter defuzzificare la variabile. Così facendo è possibile specificare regole che assegnano o utilizzano una variabile, anche contemporaneamente. Questo permette algoritmi di compiere ragionamenti a più livelli, oppure di implementare facilmente un anello di controllo in retroazione fuzzy.

5.4 Linguaggio del classificatore

Il linguaggio del classificatore è stato implementato per poter esprimere un classificatore ad albero. Tuttavia, poiché nel mondo reale, soprattutto nel campo dell'analisi di immagine, sono importanti anche le relazioni tra gli oggetti, la normale struttura ad albero è stata arricchita da archi che non rappresentano relazione di ereditarietà, ma relazioni tra le variabili di oggetti differenti.

5.4.1 Definizione delle classi

Per definire una classe si usa la keyword `CLASS`, seguita dal nome della classe e eventualmente un modificatore. La definizione comprende la lista delle variabili e delle relazioni tra classi, ed è conclusa dalla keyword `END_CLASS`.

è possibile specificare due modificatori:

extends seguito dal nome di una classe, per definire la superclasse.

HIDDEN che nasconde l'appartenenza degli oggetti alla classe in output.

Per specificare le variabili appartenenti a una classe si utilizza la keyword `VARIABLES`, seguita dal nome delle variabili utilizzate, separate dal token ";", terminate dalla keyword `END_VARIABLES`.

All'interno delle classi è anche possibile specificare delle costanti, che sono variabili con associato un fuzzy set associati. Questa caratteristica del linguaggio non è stata tuttavia ancora espansa, e le costanti possono essere utilizzate a scopo documentativo. E' possibile espandere il loro utilizzo per aumentare le capacità di riconoscimento, ad esempio controllando che l'oggetto classificato rispetti determinate costanti, oppure utilizzando le costanti per compiere ulteriori inferenze.

Non è necessario ridefinire le variabili delle superclassi nelle loro sottoclassi, le variabili infatti vengono ereditate automaticamente.

Un esempio di definizione di classe è il seguente:

```
CLASS Example extends SuperClass  HIDDEN
 VARIABLES
  var1;
  var2;
 END_VARIABLES
END_CLASS
```

5.4.2 Feature e relazioni tra classi

Per discriminare l'appartenenza di un oggetto a una determinata classe è possibile specificare alcune feature della classe.

La feature di base che è possibile esprimere è il livello di appartenenza a un determinato fuzzy set definito su una variabile di una superclasse o della classe stessa. Per esprimere questo tipo di feature, basta scrivere il nome della variabile su cui si vuole discriminare, seguita dalla keyword `is`, seguita dal nome del fuzzy set che si vuole utilizzare.

Possono essere espresse anche feature che dipendono dalla relazione di una determinata istanza della classe con altre istanze di altre classi. In particolare è possibile esprimere:

- La relazione di matching tra due variabili di due istanze; tramite la keyword `match`
- La relazione di contenimento di una variabile di un'altra istanza rispetto al valore di due variabili dell'istanza; tramite la keyword `on`
- La relazione di contenimento inversa; sempre tramite la keyword `on`

Tutti i tre tipi di relazioni possono essere qualificate, in modo da non avere il semplice contenimento o il matching esatto, ma rendere possibile un grado di matching fuzzy e di specificare in maniera più fine il contenimento, ad esempio discriminando destra/sinistra. Per farlo si utilizzano la keyword `degree` per il matching e la keyword `is` per il contenimento. Si veda la grammatica per sapere come definire tali relazioni con precisione.

Un esempio di classe contenente queste feature è il seguente:

```
CLASS Example1
 VARIABLES
  a1;
  a2;
  b;
  c;
  x;
  y;
 END_VARIABLES

 c IS FuzzySet1;
 Example2.x IS Predicate1 on (a1, a2);
 Example1.c match c degree MatchDregree;
```

```
Example3.b match b;
x is FuzzySetX on InverseRelationshpClass(a, b);
y on InverseRelationshpClass(c, d);

END_CLASS
```

5.4.3 Utilizzo della knowledgebase

Il classificatore necessita di una knowledgebase per definire il significato dei fuzzy set definiti. L'implementazione corrente necessita di definire una classe fuzzy per ogni classe del classificatore definita, col medesimo nome. Le variabili devono aver definito i fuzzy set che si intende utilizzare all'interno della classe in cui vengono definite. I fuzzy set verranno automaticamente aggiunti anche alle sottoclassi. Se si utilizzano relazioni tra oggetti qualificate, si deve definire un predicato omonimo nella classe che lo utilizza.

5.4.4 Grammatica

⟨fuzzyClassifiers⟩ ⊨ ⟨fuzzyClass⟩ ⟨fuzzyClassifiers⟩ | ⟨fuzzyClass⟩
⟨fuzzyClass⟩ ⊨ CLASS *ID* ⟨fuzzySuperclass⟩ ⟨hiddenFlag⟩ ⟨fuzzyClassElements⟩ ⟨fuzzyFeatures⟩ END_CLASS
⟨fuzzySuperclass⟩ ⊨ extends *ID* | ϵ
⟨hiddenFlag⟩ ⊨ HIDDEN | ϵ
⟨fuzzyClassElements⟩ ⊨ ⟨constants⟩ ⟨variables⟩ | ⟨variables⟩ ⟨constants⟩ | ⟨constants⟩ | ⟨variables⟩ | ϵ
⟨constants⟩ ⊨ CONSTANTS ⟨constantList⟩ ⟨END_CONSTANTS⟩
⟨constantList⟩ ⊨ ⟨var⟩ = *ID* ; ⟨constantList⟩ | ϵ
⟨variables⟩ ⊨ VARIABLES ⟨variableList⟩ END_VARIABLES
⟨variableList⟩ ⊨ ⟨var⟩ ; ⟨variableList⟩ | ϵ
⟨fuzzyFeatures⟩ ⊨ ⟨fuzzyFeature⟩ ; ⟨fuzzyFeatures⟩ | ϵ
⟨fuzzyFeature⟩ ⊨ ⟨fuzzySimpleFeature⟩ | ⟨fuzzySimpleRelation⟩ | ⟨fuzzyComplexRelation⟩ | ⟨fuzzyInverseRelation⟩
⟨fuzzySimpleFeature⟩ ⊨ ⟨var⟩ IS *ID*
⟨fuzzySimpleRelation⟩ ⊨ *ID* . ⟨var⟩ MATCH ⟨var⟩ ⟨fuzzyDegree⟩
⟨fuzzyComplexRelation⟩ ⊨ *ID* . ⟨var⟩ ⟨fuzzyConstraint⟩ ON (⟨var⟩ , ⟨var⟩)
⟨fuzzyInverseRelation⟩ ⊨ ⟨var⟩ ⟨fuzzyConstraint⟩ ON *ID* (⟨var⟩ , ⟨var⟩)
⟨fuzzyConstraint⟩ ⊨ IS *ID* | ϵ
⟨fuzzyDegree⟩ ⊨ DEGREE *ID* | ϵ
⟨var⟩ ⊨ *ID* | *VAR_ID*

5.5 Classificazione

L'algoritmo di classificazione si può riassumere nei seguenti passaggi:

1. Costruzione del grafo delle dipendenze.
2. Generazione delle regole fuzzy definite dal classificatore.

3. Costruzione del grafo di reasoning.

4. Creazione dell'albero di reasoning.

5. Classificazione delle istanze.

6. Eliminazione delle istanze irrilevanti.

I primi tre passi dell'algoritmo sono effettuati in fase di inizializzazione, quando un classificatore fuzzy viene caricato. Gli ultimi tre passi vengono effettuati ogni volta che si deve classificare una serie di istanze. I passi 4/5 vengono ripetuti iterativamente fino ad aver esplorato tutto il grafo di reasoning. L'ultimo passo viene effettuato prima di restituire in output i risultati della classificazione.

5.5.1 Generazione delle regole fuzzy

Le regole fuzzy vengono costruite in maniera automatica a partire dal classificatore definito e dalla relativa knowledgebase. Per ogni classe, viene generata una regola che contiene la congiunzione di tutte le possibili feature che definiscono la classe, e in output una variabile, con lo stesso nome della classe, a cui viene assegnata il fuzzy set $True, che è un singleton il cui valore è 1. Questa variabile rappresenta il grado di appartenenza delle singole istanze alla classe considerata.

Le variabili che compaiono all'interno della regola sono di due tipi: variabili già presenti nella definizione della classe, e quindi aventi un nome definito dall'utente, e variabili generate automaticamente dalle relazioni tra le classi. Queste ultime variabili non sono già presenti in input, ma devono essere calcolate dal classificatore. È quindi necessario salvare i dati necessari a calcolarle quando verranno classificate le istanze e le loro dipendenze.

Il nostro algoritmo inoltre, si assicura di estendere le variabili definite nelle superclassi alle sottoclassi.

5.5.2 Grafo delle dipendenze e grafo di reasoning

Il grafo delle dipendenze è costruito a partire dalla struttura del classificatore. Vengono distinti due tipi di dipendenze, quelle di superclasse e quelle di relazione tra oggetti. Il grafo può contenere autoanelli, infatti potrebbero essere definite relazioni tra istanze della stessa classe, e cicli, infatti nulla vieta di definire dipendenze cicliche tra gli oggetti.

Sia gli autoanelli che i cicli sono problematici per il reasoning, perché per definire se un'istanza appartiene a una determinata classe, è necessario

conoscere istanze di altre classi, che a loro volta dipendono dal fatto che sia necessario conoscere l'istanza della classe di partenza. Per risolvere questo problema abbiamo sviluppato il grafo di reasoning, che permette di risolvere le dipendenze cicliche analizzando la struttura del grafo delle dipendenze.

Il grafo di reasoning è ricavato dal grafo delle dipendenze trovando tutte le componenti fortemente connesse, ossia i sottografi in cui ogni nodo è raggiungibile da ogni altro nodo. L'algoritmo per trovare le componenti fortemente connesse è descritto in [40]. Le componenti fortemente connesse contengono tutti i cicli del grafo. Questo può essere dimostrato facilmente nel seguente modo:

Dimostrazione. Sia preso un grafo G con N componenti connesse. Supponiamo per assurdo che esista un ciclo tra elementi che appartengono a diverse componenti fortemente connesse. Allora esiste un insieme di nodi I, tutti raggiungibili tra loro a causa dell'appartenenza a un ciclo, che non appartengono alla stessa componente fortemente connessa. Assurdo. □

Il grafo di reasoning è definito nel seguente modo:

- I nodi del grafo di reasoning sono le componenti connesse del grafo delle dipendenze.
- Esiste una connessione tra due nodi c_1 e c_2 del grafo di reasoning se e solo se esiste almeno un collegamento diretto tra un nodo i, appartenente alla componente fortemente connessa c_1 e un nodo j, appartenente alla componente fortemente connessa c_2 e $c_1 \neq c_2$, dove i e j sono nodi del grafo delle dipendenze.

Dimostriamo l'aciclicità del grafo di reasoning come segue:

Dimostrazione. Sia R un grafo di reasoning ricavato dal grafo delle dipendenze D. Supponiamo per assurdo che il grafo R sia ciclico. Allora esistono almeno due nodi del grafo R c_1 e c_2, con $c_1 \neq c2$ che sono mutuamente raggiungibili tra di loro. Allora esiste almeno un nodo $i \in D$ in c_1 con un arco entrante in un nodo di c_2, e un nodo $j \in D$ in c_2 con un arco entrante in un nodo di c_1. Poiché sia c_1 sia c_2 sono componenti fortemente connesse, e per transitività della relazione di raggiungibilità, ogni nodo di c_1 può raggiungere ogni nodo di c_2 e viceversa. quindi $c_1 = c_2$. Assurdo. □

E' possibile dunque imporre un ordinamento topologico [18] sul grafo, ed eseguire la classificazione delle istanze nell'ordine specificato, analizzando un cluster di componenti fortemente connesse alla volta. Per ogni cluster è

necessario testare ogni possibile combinazione di istanze candidate a appartenere a ciascuna delle classi al suo interno. Poiché gli archi delle dipendenze vanno da sottoclasse a superclasse e da classe a dipendenza, l'ordinamento topologico preso è quello inverso.

5.5.3 Classificazione delle istanze

Seguendo uno dei possibili ordinamenti topologici imposti dal grafo di reasoning, Vengono classificate le istanze di ogni classe.

La classificazione avviene seguendo i seguenti passi:

1. Per ogni classe contenuta in un cluster di componenti connesse, si estraggono tutti i possibili candidati. Un candidato ha tutte le variabili della classe e appartiene alla eventuale superclasse.
2. Si enumerano tutte le possibili combinazioni di istanze nel cluster.
3. Per ogni combinazione di istanze, si enumerano le possibili dipendenze di ogni istanza all'esterno del cluster.
4. Per ogni combinazione di istanze e dipendenze si calcolano le variabili che dipendono dalle relazioni tra classi.
5. Tramite le variabili precedentemente note e quelle calcolate, si calcola il grado di appartenenza alla classe dell'istanza.
6. Per ogni istanza, si prende come grado di appartenenza il minimo tra il valore calcolato e il valore di verità della superclasse.
7. Si calcola il valore di verità minimo tra la corrente combinazione di cluster.
8. Se il grado di appartenenza è maggiore di zero e maggiore uguale alla soglia specificata come input dell'algoritmo, si assegna il grado di appartenenza alle istanze.
9. A ogni istanza viene assegnato come grado di appartenenza alla classe il massimo tra un eventuale grado già assegnato, e il corrente grado calcolato.

Per implementare questo algoritmo in pratica, abbiamo scelto di esplorare l'albero delle possibili combinazioni, e di classificare le istanze una volta raggiunte le singole foglie. le classificazioni sono salvate i una mappa accessibile dall'algoritmo, e quindi in ogni nodo si hanno tutti i dati necessari alla classificazione.

Una volta completata la classificazione di un cluster, si passa alla classificazione del cluster successivo, fino a terminare tutti i cluster. Poiché il grafo di reasoning è esplorato seguendo l'ordinamento topologico, tutte le superclassi e le dipendenze esterne di un cluster vengono analizzate prima del cluster stesso.

Finito di esplorare il grafo delle dipendenze, tutte le classi etichettate con la keyword `HIDDEN` vengono cancellate dall'output calcolato. Questo permette di creare classi strutturali per il classificatore, particolarmente utili se il classificatore è totalmente o parzialmente appreso tramite tecniche di machine learning, ma utilizzabili anche per definire classi comuni di feature e rendere più efficiente il riconoscimento, riducendo le feature da classificare all'interno di cluster complessi. Inoltre, le classi nascoste sono una caratteristica della quasi totalità dei classificatori fuzzy ad albero presenti nella letteratura [43], dove solitamente solo le foglie sono visibili in output, come nei classici alberi di decisione.

Capitolo 6

Riconoscimento degli Oggetti

Marty: Ok, rilassati Doc, sono io, sono io Marty!
Doc: No, non può essere . . . io ti ho rimandato nel futuro!
Marty: Lo so mi hai rimandato indietro nel futuro, ma sono tornato, sono tornato dal futuro!
Doc: Grande Giove!

Ritorno al Futuro, parte II

6.1 Introduzione

In questo capitolo descriveremo il riconoscimento di oggetti. Il riconoscimento degli oggetti si basa sull'estrazione di feature geometriche e la loro classificazione tramite un classificatore fuzzy ad albero, che sfrutta il reasoner descritto nel Capitolo 5. Il sistema qui descritto si pone l'obbiettivo di dimostrare una possibile applicazione del sistema di classificazione basato sulla conoscenza di un esperto, delineando le metodologie per estrarre feature e classificarle. Tuttavia un sistema completo di classificazione delle feature può estrarre molta più informazione dall'immagine, sia come quantità di feature estraibili (ad esempio estraendo coniche e piani), sia come informazioni caratterizzanti (ad esempio aggiungendo il colore).

Il sistema implementato è pensato principalmente per estrarre quadrilateri e cluster, e analizzarli. Questa scelta è stata fatta perché la maggior parte degli oggetti presenti in un ambiente, come una stanza o un ufficio, si compone di oggetti planari rettangolari (porte, finestre, armadi). Questi oggetti spesso sono caratterizzati da componenti più piccole, come ad esempio le maniglie, che hanno la caratteristica di avere una geometria complessa. Questo causa una massiccia concentrazione di keypoint, rendendo possibile la loro individuazione tramite algoritmi di clustering.

Il riconoscimento avviene su più livelli: prima vengono analizzate le feature più facili da riconoscere. In seguito per ogni feature riconosciuta viene effettuata una analisi ad alto livello.

6.2 Individuazione delle feature geometriche

Le feature geometriche sono individuate da algoritmi specifici. Tuttavia è stata definita una struttura di base, rappresentata dalla classe feature. La classe feature implementa l'interfaccia tra il classificatore e l'algoritmo di riconoscimento nell'immagine. La Figura 6.1 mostra la struttura delle classi del sistema implementato. Come si nota, la classe feature implementa due strutture dati:

FeatureMap è un dizionario che contiene i dettagli dell'oggetto riconosciuto, con il nome delle feature e il loro valore.

ClassificationMap è un dizionario che contiene tutte le classificazioni attribuite all'oggetto, con il relativo grado di appartenenza.

La FeatureMap è popolata tramite il metodo setFeature a partire dai dati caratteristici della stessa. La ClassificationMap è popolata tramite il metodo addClassification. La classe feature contiene ulteriori metodi per l'analisi semantica dell'oggetto.

Di seguito vengono discussi gli algoritmi implementati per il riconoscimento delle feature geometriche. La base di partenza per tutti gli algoritmi considerati è l'immagine rettificata della telecamera, ossia l'immagine senza la distorsione causata dalle caratteristiche costruttive della lente, ovvero la distorsione radiale e tangenziale.

6.2.1 Quadrilateri

L'algoritmo di riconoscimento di quadrilateri si basa principalmente sull'algoritmo di Canny [5] per il riconoscimento dei bordi e sulla trasformata di Hough probabilistica [29] per estrarre le linee, o più precisamente, i segmenti, dalla mappa dei bordi.

Le linee estratte contengono, quando l'ambiente è minimamente complesso, molte linee spurie. Inoltre, i quadrilateri interessanti sono composti da due linee approssimativamente orizzontali e due linee approssimativamente verticali. Per questo è stato implementato un algoritmo di filtraggio delle linee in grado di dividere le linee orizzontali da quelle verticali e dal rumore.

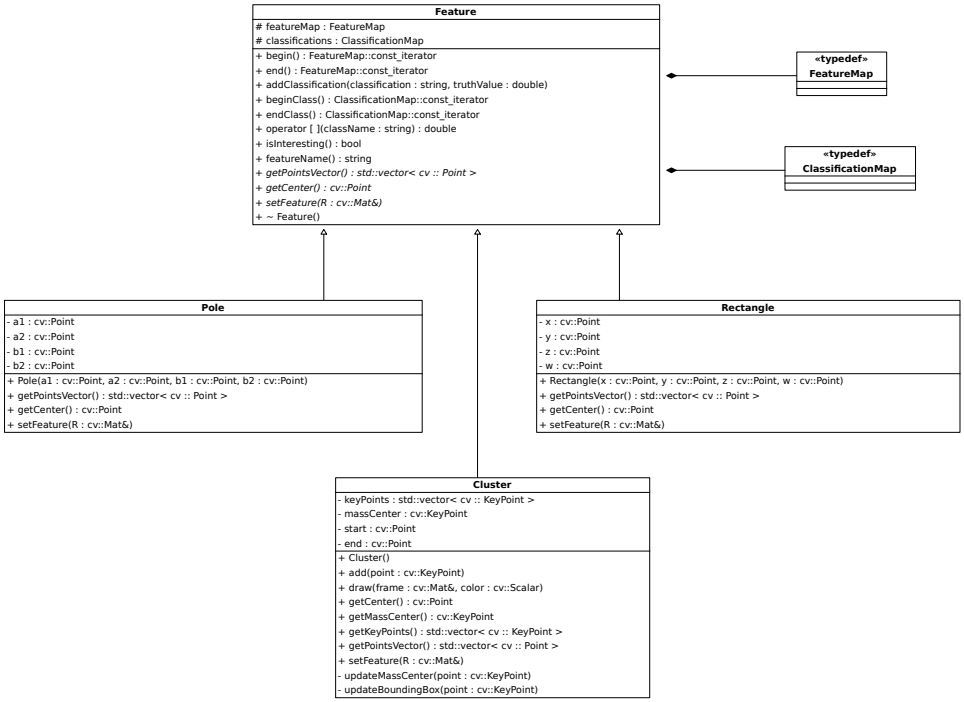

Figura 6.1: Classe Feature e sottoclassi implementate

Per discriminare le linee si utilizzano i dati dell'odometria del robot. Conoscendo infatti la posa del robot approssimativamente, data l'odometria, è possibile stimare l'inclinazione dell'orizzonte e della linea perpendicolare ad esso. Le linee orizzontali e verticali sono quelle la cui inclinazione, espressa in angolo rispetto all'asse delle ascisse, non supera una determinata soglia rispetto all'orizzonte e alla sua perpendicolare.

Una volta riconosciute le linee verticali e orizzontali viene applicato l'Algoritmo 1 per riconoscere quadrilateri e pali. Viene considerato palo una coppia di linee con un fattore di forma estremamente schiacciato verticalmente. Nell'algoritmo descritto V rappresenta il vettore di linee verticali (ordinate da destra a sinistra), H l'insieme delle linee orizzontali, Q l'insieme di quadrilateri trovati.

Questo algoritmo si basa su una considerazione fondamentale per ridurre la complessità del calcolo, ossia che è altamente probabile che un rettangolo interessante si trovi tra due linee verticali consecutive. Questa considerazione è sostenuta dal fattore di forma dell'immagine, che è largo, e dal fattore di forma della maggior parte dei quadrilateri interessanti, che solitamente sono più stretti che larghi. Questa considerazione non si può tuttavia estendere al caso delle linee orizzontali, per le motivazioni opposte, e per questo

Algorithm 1 QuadrilateralDetector

```
1: for i ∈ {0, VerticalLinesNumber − 1} do
2:    v1 ← V[i]
3:    v2 ← V[i + 1]
4:    if v1 and v2 are not poles then
5:       for all (h1, h2) ∈ H × H do
6:          x ← findInterception(h1, v1);
7:          y ← findInterception(h1, v2);
8:          z ← findInterception(h2, v2);
9:          w ← findInterception(h2, v1);
10:         if x, y, z, w are the vertices of a quadrilateral q then
11:            Q = Q ∪ {q}
12:         end if
13:      end for
14:   end if
15: end for
16: return Q
```

la ricerca è fatta in maniera esaustiva.

Per discriminare se un insieme di punti formano o meno un quadrilatero vengono utilizzati due euristiche, a seconda del riconoscimento effettuato. L'euristica più semplice consiste nel considerare tutto ciò che non è un palo come un quadrilatero. Questa euristica in generale è pessima, ma funziona molto bene, avendo un tasso di recupero decisamente alto, quando il riconoscimento è localizzato.

L'altra euristica possibile che abbiamo sviluppato consiste nello sfruttare le proprietà delle combinazioni convesse. Una combinazione convessa è definita come una combinazione lineare con coefficienti positivi e la cui somma vale 1. L'insieme delle combinazioni convesse di un insieme di punti generano l'involucro convesso dell'insieme. Nel nostro caso, preso un segmento riconosciuto come possibile lato del quadrilatero, possiamo calcolare qualsiasi punto al suo interno tramite la formula:

$$x = \alpha \cdot a + (1 - \alpha) \cdot b = b + \alpha \cdot (a - b) \tag{6.1}$$

Dove a e b sono gli estremi del segmento e x è un punto al suo interno, i coefficienti di combinazione convessa sono α e $1 - \alpha$. Come si nota dalla formula $\alpha = 0$ se $x \equiv b$ e $\alpha = 1$ se $x \equiv a$. Idealmente, per i vertici di un quadrato, si avrà $\alpha = 1$ o $\alpha = 0$ se si calcola il coefficiente del vertice rispetto ai due segmenti adiacenti.

Se si prova a calcolare il valore di α con la stessa formula per un punto fuori dal segmento si ottiene un valore maggiore di uno o minore di zero.

L'euristica proposta si basa sulle considerazioni fatte sopra, supponendo che a causa del rumore e altri fattori, i vertici del quadrato riconosciuto si possano trovare all'interno o all'esterno del segmento. Viene quindi calcolato il valore di combinazione convessa di ciascun vertice e, fissata una soglia, viene riconosciuto come quadrato solo un insieme di segmenti i cui vertici hanno $\alpha = 1$ o $\alpha = 0$, a meno della soglia.

6.2.2 Cluster

I cluster vengono riconosciuti estraendo i keypoint dall'immagine tramite l'algoritmo FAST [37].

Una volta estratti i keypoint, vengono estratti i cluster interessanti tramite l'algoritmo DBSCAN [8]. DBSCAN è un algoritmo che si basa sulla distanza tra i punti. Due punti vengono considerati vicini se la distanza tra di loro è minore di una determinata soglia. Vengono quindi inseriti in un unico cluster tutti i punti che sono raggiungibili tramite la relazione di vicinanza. Ciò significa che tutti i vicini di un punto sono inseriti nello stesso cluster, e allo stesso modo i vicini dei vicini, ricorsivamente. I cluster che non hanno abbastanza punti al loro interno vengono scartati e considerati come rumore.

6.3 Integrazione con il reasoning

Una volta riconosciute le feature interessanti, vengono calcolati i descrittori di tali feature. I descrittori delle feature vengono inseriti nella FeatureMap, per poter comunicare la feature riconosciuta al classificatore fuzzy. Viene effettuata la rotazione delle coordinate in modo da avere i quadrilateri riconosciuti il più possibile orientati verticalmente. Questo facilita il compito del reasoner, in quanto attualmente l'operatore "on" supporta solo intervalli semplici e non aree o intervalli generici (si veda Capitolo 5). Le richieste di classificazione vengono, per quanto possibile, inviate contemporaneamente al reasoner, in modo da rendere possibile l'analisi delle relazioni tra gli oggetti riconosciuti. Viene poi popolata, con i dati provenienti dal reasoner, la ClassificationMap. Questo è reso possibile perché ogni feature inviata è associata a un indice univoco.

6.4 Livelli di riconoscimento

Il riconoscimento degli oggetti non è effettuato tramite una singola analisi dell'immagine, ma viene effettuato su più livelli, sfruttando tutte le informazioni possibili.

Il riconoscimento di basso livello estrae possibili feature dall'immagine intera, in modo che le feature riconosciute possano essere seguite dal tracker. Poiché è importante escludere i falsi positivi il più possibile, e poiché è più complesso analizzare una intera immagine senza conoscere a priori la posizione degli oggetti interessanti, il riconoscimento di basso livello è molto restrittivo. L'euristica utilizzata per riconoscere i quadrati nell'immagine è quella che sfrutta le proprietà della combinazione convessa. Per rendere la computazione più semplice e rapida, non vengono estratti cluster dall'immagine, e le uniche feature analizzate dal reasoner sono i quadrilateri che rappresentano possibili rettangoli.

Una volta riconosciuti i possibili rettangoli interessanti, gli oggetti vengono quindi seguiti dal tracker. Grazie alle informazioni del tracker, descritte nel Capitolo 7, è possibile avere un'analisi degli oggetti più localizzata. Questo permette quindi un'analisi dell'immagine più semplice, ed è quindi possibile utilizzare tecniche meno discriminative e più robuste. Usiamo quindi l'euristica più semplice (tutto è un quadrilatero) e estraiamo cluster solo in prossimità della regione di interesse, in modo da risparmiare potenza computazionale sia durante l'estrazione delle feature, sia durante il reasoning, eliminando gli oggetti incorrelati. Questa ultima fase quindi è in grado di analizzare feature complesse come gli oggetti composti da più parti, ad esempio le porte.

Capitolo 7

Tracking

Marty: Ma allora dove diavolo sono?
Doc: La domanda giusta è: 'Quando diavolo sono?'

Ritorno al Futuro, parte I

7.1 Introduzione

Una volta riconosciute le possibili feature, è necessario seguirle nelle immagini successive della telecamera. Per farlo ci siamo basati su un algoritmo presente in letteratura, chiamato Consensus-based Matching and Tracking, CMT [31]. L'algoritmo considerato è un algoritmo di tracking a lungo termine. Gli algoritmi di tracking a lungo termine permettono di inseguire oggetti che entrano e escono dal campo visivo della telecamera, e cercano di discriminare anche oggetti simili.

Il tracker implementato è in grado non solo di riconoscere gli oggetti nell'immagine, ma è in grado, in parte, di capire se gli oggetti riconosciuti dall'algoritmo descritto nel Capitolo 6 sono già o meno nella lista delle track.

In questo capitolo spiegheremo brevemente come funziona l'algoritmo CMT utilizzato e come è stato integrato con il riconoscimento di oggetti e con la costruzione della mappa.

7.2 CMT

L'idea base dell'algoritmo è che i keypoint sono un buon modo per scomporre in parti l'oggetto da seguire. Il primo passo consiste quindi nell'estrarre keypoint dal bounding box iniziale, calcolare i relativi descrittori delle feature e salvare tutto in un database. Questa idea è resa possibile dal recente

sviluppo di algoritmi veloci ed efficienti per l'estrazione di feature quali [37] e [24].

Successivamente, per ogni frame, i keypoint presenti nell'iterazione precedente vengono inseguiti tramite il calcolo del flusso ottico [26] di Lukas-Kanade, nella variante piramidale [44]. Inoltre vengono estratti keypoint e effettuato il matching con i keypoint presenti nel database. I keypoint di cui è stato effettuato con successo il matching vengono sostituiti a quelli seguiti tramite il flusso ottico, poiché si suppone che siano più stabili, non essendo parte di un processo iterativo di valutazione, soggetto ad errore integrale.

La seconda fase consiste in una procedura di votazione per determinare il centro di massa dell'oggetto. In primo luogo vengono stimate la scala e la rotazione planare dell'oggetto, calcolando la variazione di scala e di rotazione tra tutte le coppie di keypoint e calcolandone la mediana. In seguito ogni keypoint vota il centro di massa dell'oggetto utilizzando la sua posizione relativa al centro di massa nel primo frame, opportunamente scalato e ruotato grazie ai due valori precedentemente calcolati.

La terza fase dell'algoritmo consiste nell'eliminazione degli outlier tramite un meccanismo di consenso. I centri di massa votati da ciascun singolo keypoint vengono divisi in cluster in base alla distanza euclidea. Quindi, viene individuato il cluster con il maggior numero di voti. Se il cluster scelto ha un numero sufficiente di keypoint, allora l'oggetto viene considerato come visibile e viene calcolato il centro di massa dell'oggetto, eliminando tutti i voti che non appartengono al cluster.

Infine il bounding box dell'oggetto viene calcolato applicando la rotazione e la scala calcolate precedentemente ai punti del bounding box iniziale. Questo procedimento è il punto più debole dell'algoritmo, perché il bounding box calcolato non tiene conto dell'omografia che può esserci tra il bounding box iniziale e quello attuale, causata, ad esempio, dalla distorsione prospettica a seguito della nuova posa della telecamera.

La nostra implementazione si discosta dall'implementazione originale, su cui è fortemente basata, solo per la possibilità di specificare un bounding box con una forma arbitraria.

7.3 Integrazione con il riconoscimento degli oggetti

L'algoritmo di tracking deve essere in grado di interagire con il riconoscimento a basso e alto livello degli oggetti. Per ottenere questo risultato ab-

biamo implementato alcune euristiche per gestire a basso livello e in maniera semplice i possibili conflitti tra l'algoritmo di tracking e il riconoscimento.

In primo luogo, per facilitare sia il tracking dell'oggetto, sia il riconoscimento ad alto livello, quando il tracker considera un nuovo oggetto da seguire inviato dal riconoscimento a basso livello, non utilizza il bounding box esatto riconosciuto, ma viene maggiorato, scalandolo, in modo da ottenere due importanti risultati:

1. Riuscire a riconoscere in maniera efficiente i keypoint che si trovano sui bordi degli oggetti, che spesso sono i keypoint più significativi, soprattutto negli oggetti planari e uniformi, principali obbiettivi di questa tesi.

2. Riuscire a compensare parzialmente il problema causato dal cambio di posa della telecamera, che potrebbe causare la fuoriuscita dell'oggetto, o parte di esso, dal bounding box, a causa della deformazione prospettica, non calcolata dall'algoritmo.

Le track inviate al riconoscimento di alto livello comprendono due informazioni: il bounding box (maggiorato) e la regione di interesse rettangolare che lo contiene. Questo permette al riconoscimento di alto livello di analizzare solo le regioni di interesse rettangolari dell'immagine, rendendo il compito meno oneroso computazionalmente e di cancellare eventuali feature esterne all'oggetto, che non sono di interesse per il riconoscimento.

L'ultimo problema riguardante l'integrazione, il più problematico, è riuscire a discriminare quali oggetti riconosciuti sono già presenti nella lista delle track.

Questo problema può essere risolto a due livelli. Ad alto livello, è possibile capire che due oggetti trackati nella mappa occupano lo stesso spazio, e quindi fanno parte dello stesso oggetto. Questa soluzione è la soluzione ottimale, ma richiede un maggior numero di informazioni e un costo computazionale maggiore. Tuttavia è possibile affrontare parzialmente il problema a basso livello, escludendo in maniera semplice gli oggetti riconosciuti. L'euristica che abbiamo sviluppato si basa sul calcolo della posizione del centro di massa degli oggetti riconosciuti e delle track, rispetto ai loro bounding box. Se un oggetto riconosciuto ha il suo centro di massa all'interno di un bounding box di una track attiva, o, viceversa, se una track attiva ha il suo centro di massa nel bounding box dell'oggetto riconosciuto, tale oggetto è considerato già seguito, e non viene aggiunto alla lista delle track.

Come si può vedere dalla figura, l'euristica copre abbastanza bene quasi tutti i casi, a parte quando si tratta di oggetti composti da più oggetti

o oggetti che si trovano di fronte ad altri oggetti. Per questi casi sono necessarie due ragionamenti ad alto livello differenti: riuscire a capire che un oggetto è semanticamente composto da alcune parti; riuscire a capire che i due oggetti non occupano lo stesso spazio. Per entrambi i ragionamenti non bastano le informazioni estratte dalla singola immagine.

7.4 Integrazione con la mappa

L'euristica descritta nel paragrafo precedente dimostra comportarsi molto bene nelle situazioni reali, tuttavia subentrano alcuni problemi quando il tracking degli oggetti presenta dei mancati riconoscimenti. Questo causa spesso la sovrapposizione di più track dello stesso oggetto. Questo problema si può risolvere solamente integrando l'algoritmo di tracking con la mappa degli oggetti riconosciuti. Tramite la mappa è facile accorgersi che lo stesso oggetto è seguito più volte, poiché occupa la stessa posizione dell'oggetto seguito in precedenza.

Il tracking degli oggetti è un compito abbastanza oneroso computazionalmente. Se la posizione del robot è nota, anche in maniera imprecisa, non è necessario mantenere attivo l'algoritmo di tracking per tutti gli oggetti. Inoltre l'algoritmo usato è soggetto a falsi positivi; potendo filtrare le track che non possono essere attive, perché non si trovano nell'area vista dal robot in un determinato istante, il rate di falsi positivi cala drasticamente.

Infine grazie alla stima della posizione, si può calcolare l'omografia che tiene conto della deformazione prospettica del bounding box dell'oggetto rispetto al nuovo punto di vista. Così facendo si può facilitare il riconoscimento ad alto livello.

Capitolo 8

Mapping

Capitolo 9

Risultati sperimentali

Capitolo 10

Conclusioni e Sviluppi Futuri

Bibliografia

[1] Yotam Abramson, Bruno Steux, and Hicham Ghorayeb. Yet even faster (yef) real-time object detection. *IJISTA*, 2(2/3):102–112, 2007.

[2] S. Blumenthal, H. Bruyninckx, W. Nowak, and E. Prassler. A scene graph based shared 3d world model for robotic applications. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 453–460, May 2013.

[3] Sebastian Blumenthal and Herman Bruyninckx. Towards a domain specific language for a scene graph based robotic world model. *arXiv preprint arXiv:1408.0200*, 2014.

[4] Bruce G Buchanan, Edward Hance Shortliffe, et al. *Rule-based expert systems*, volume 3. Addison-Wesley Reading, MA, 1984.

[5] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, Nov 1986.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, June 2005.

[7] Randall Davis, Bruce Buchanan, and Edward Shortliffe. Production rules as a representation for a knowledge-based consultation program. *Artificial intelligence*, 8(1):15–45, 1977.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[9] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based mo-

dels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[10] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[11] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[12] M Herbert, C Caillas, Eric Krotkov, In So Kweon, and Takeo Kanade. Terrain mapping for a roving planetary explorer. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 997–1002. IEEE, 1989.

[13] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-primer/`.

[14] Helmi Ben Hmida, Christophe Cruz, Frank Boochs, and Christophe Nicolle. Knowledge base approach for 3d objects detection in point clouds using 3d processing and specialists knowledge. *CoRR*, abs/1301.4991, 2013.

[15] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[16] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[17] Gary D. Riley Joseph C. Giarratano. *Expert Systems Principles and Programming*. 1994.

[18] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, November 1962.

[19] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[20] Georg Klein and David Murray. Improving the agility of keyframe-based SLAM. In *Proc. 10th European Conference on Computer Vision (ECCV'08)*, pages 802–815, Marseille, October 2008.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[22] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Detection-based object labeling in 3d scenes. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1330–1337. IEEE, 2012.

[23] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 1442–1447 vol.3, Nov 1991.

[24] S. Leutenegger, M. Chli, and R.Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555, Nov 2011.

[25] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002*, pages 900–903, 2002.

[26] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[27] Nicolas Maillot and Monique Thonnat. Ontology based complex object recognition. *Image Vision Comput.*, 26(1):102–113, 2008.

[28] Ebrahim H Mamdani and Sedrak Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1):1–13, 1975.

[29] Jiri Matas, Charles Galambos, and Josef Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78(1):119–137, 2000.

[30] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[31] Georg Nebehay and Roman Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision*. IEEE, March 2014.

[32] Richard A. Newcombe, Steven Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 2320–2327. IEEE, 2011.

[33] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[34] C.P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Computer Vision, 1998. Sixth International Conference on*, pages 555–562, Jan 1998.

[35] Patrick Pfaff, Rudolph Triebel, and Wolfram Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *The International Journal of Robotics Research*, 26(2):217–230, 2007.

[36] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[37] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[38] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, June 1989.

[39] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[40] Robert Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1972.

[41] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2276–2282, Oct 2006.

[42] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[43] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69(2):125 – 139, 1995.

[44] Jean yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[45] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

Appendice A

Il manuale utente

A.1 Installazione e compilazione

Per compilare il sistema in maniera completa è necessaria una istallazione completa di ROS. Si veda la guida ufficiale di ROS per conoscere i dettagli dell'istallazione del sistema. È necessario utilizzare ROS Hydro Medusa o superiore. Inoltre è necessario installare i generatori di parser `Flex` e `Bison`. È possibile compilare la libreria che implementa l'algoritmo di reasoning anche al di fuori di ROS, utilizzando semplicemente `CMake`. In tal caso si raccomanda di installare anche le librerie `Boost`.

Una volta installate le dipendenze necessarie, è possibile compilare i sorgenti. Per farlo, basta creare un workspace di catkin, aggiungere i package nella cartella src, e lanciare catkin_make. Le compilazione avverrà nell'ordine corretto, rispettando le dipendenze tra i package. È necessario aggiungere il file setup.bash nella cartella `devel/` del workspace come script da eseguire nel vostro file .bashrc; ad esempio, se il workspace si chiama catkin_ws, e si trova nella home, basterà aggiungere al file .bashrc la linea:

```
source ~/catkin_ws/devel/setup.bash
```

Per lanciare i singoli nodi si può utilizzare il programma rosrun. Ricordarsi di aggiungere gli argomenti da linea di comando e i parametri privati necessari subito dopo il nome del package e del nodo che si vuole lanciare. I parametri privati vengono specificati aggiungendo un "_" prima del nome del parametro; il cui valore viene assegnato con il token ":=".

Ad esempio per lanciare il nodo c_fuzzy_reasoner nel package c_fuzzy si può utilizzare il comando:

```
rosrun c_fuzzy c_fuzzy_reasoner -c knowledgebase.kb classifier.fuzzy
```

Una alternativa per lanciare tutti i nodi contemporaneamente è usare un launchfile. Launchfile di esempio sono presenti nel package c_slam.

Per lanciare l'intero sistema si può utilizzare il comando:

```
roslaunch c_slam c_slam.launch
```

A.2 Parametri e argomenti

Di seguito sono elencati e spiegati i parametri utilizzati dai nodi. si veda il Capitolo 3 per informazioni riguardo agli algoritmi.

A.2.1 c_vision_detector e c_vision_slam

Questi nodi hanno bisogno necessariamente di parametri privati, senza i quali il loro funzionamento non è garantito.

Canny

Questi parametri sono utilizzati dall'algoritmo Canny Edge detector.

canny/alpha rappresenta il valore della soglia bassa, in proporzione alla soglia alta.

canny/apertureSize rappresenta la dimensione del kernel con il quale applicare l'operatore di Sobel. Deve essere necessariamente un intero dispari e maggiore o uguale a 3.

Hough

Questi parametri sono utilizzati per l'algoritmo di riconoscimento delle linee, detto trasformata di Hough probabilistica.

hough/rho risoluzione della distanza dall'origine delle rette espressa in pixel.

hough/teta risoluzione dell'inclinazione delle rette, espressa in gradi.

hough/threshold threshold per filtrare le linee dal rumore.

hough/minLineLenght minima lunghezza delle linee riconosciute, espressa in pixel.

hough/maxLineGap massima distanza tra due punti appartenenti alla stessa linea, in pixel.

Line filtering

Questi parametri vengono utilizzati dall'algoritmo che distingue le linee orizzontali e verticali dal rumore.

filter/maxDeltaHorizontal massima inclinazione delle rette orizzontali rispetto alla linea dell'orizzonte, in gradi.

filter/maxDeltaVertical massima inclinazione delle rette verticali rispetto alla linea perpendicolare all'orizzonte, in gradi.

Clustering

Questi parametri servono per tarare il riconoscimento dei cluster. Sono utilizzati dagli algoritmi FAST, per riconoscere i keypoints, e DBSCAN per estrarre da essi i cluster.

cluster/threshold threshold per l'estrazione di keypoints dall'immagine.

cluster/minPoints numero minimo di punti necessari a definire un cluster.

cluster/maxDistance massima distanza tra i punti di un cluster.

Questi parametri non sono necessari nel nodo c_vision_detector.

classifier

Questi parametri sono utilizzati dai nodi di visione quando vengono preparate le richieste di reasoning, e quindi sono di fatto utilizzati nel classificatore fuzzy.

classifier/threshold threshold da utilizzare nella classificazione delle feature.

A.2.2 c_fuzzy_reasoner

Per lanciare il reasoner è necessario specificare dei parametri da linea di comando.

-h stampa il messaggio di aiuto

-r knowledgebase crea il servizio di reasoning basato sulla knowledgebase specificata.

-c knowledgebase classifier crea il servizio di classificazione a partire dalla knowledgebase e dal classificatore specificati.

Appendice B

Esempio di impiego

Un esempio di impiego del sistema realizzato.