SAPIENZA
UNIVERSITÀ DI ROMA

Gestione di reception mediante robot mobili

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Ingegneria Informatica e Automatica

Giovanni Magliocchetti
Matricola 1896778

Relatore
Prof. Luca Iocchi

Anno Accademico 2021/2022

Gestione di reception mediante robot mobili
Sapienza Università di Roma

Questa relazione è stata composta con LaTeX e la classe Sapthesis, utilizzando un'interlinea di 1.5 ed una dimensione carattere di 12pt.

Email dell'autore: magliocchetti.1896778@studenti.uniroma1.it

Sommario

Questa relazione tratta della programmazione di robot mobili per la realizzazione del Task numero 3 "Receptionist" della RoboCup@Home Education Challenge 2022. In sette capitoli: introduzione della competizione, presentazione del robot, spiegazione del task e dei problemi affrontati, implementazione della soluzione, riflessioni su possibili migliorie e conclusioni.

Indice

Capitolo 1

Introduzione

Questa relazione parlerà del tirocinio svolto con il Professor Luca Iocchi del Dipartimento di Ingegneria Informatica, Automatica e Gestionale (DIAG). L'argomento trattato è la programmazione di robot mobili ed in particolare è stato scelto un task dalla RoboCup@Home Education Challenge 2022. L'obbiettivo è stato quello di programmare un robot affinché soddisfacesse i requisiti per partecipare a questa competizione, attraverso lo svolgimento di uno dei tre task previsti dal Rulebook 2022. Il robot mi è stato fornito dal DIAG e dalla Robotics 3D, dunque non mi sono dovuto occupare né dell'assemblaggio né della scelta dei componenti. Per quanto riguarda la parte software, il robot è basato su MARRtino, una piattaforma basata su ROS, che mi ha permesso di programmarlo ad alto livello.

Nel Capitolo 2 si parla della RoboCup@Home Education Challenge 2022

Nel Capitolo 3 si espone la struttura di MARRtino

Nel Capitolo 4 si descrive il task scelto ed i metodi utilizzati per risolverlo

Nel Capitolo 5 si tratta l'implementazione del task

Nel Capitolo 6 si riflette su possibili migliorie

Capitolo 2

RoboCup@Home Education Challenge 2022

La RoboCup@Home Education Challenge è una piattaforma di competizione educativa per coltivare squadre di principianti per le sfide di RoboCup@Home e lo sviluppo di robot di servizio. Lo scopo della Sfida Educativa è aprire la partecipazione a tutti, non solo agli studenti universitari tecnici, ma anche agli studenti delle scuole superiori inesperti e al grande pubblico. L'esclusivo formato Workshop + Competizione stimola efficacemente i partecipanti alle prime armi per lo sviluppo di robot di servizio impegnativi e l'apprendimento dell'Intelligenza Artificiale durante il tempo dell'evento. Essa viene ospitata a livello locale (regionale e nazionale) ed internazionale.

Figura 2.1. RoboCup@Home Education Challenge @ RoboCup Japan Open 2019.

2.1 Tasks

La challenge propone tre diversi task tra i quali è possibile scegliere quello che si vuole realizzare. I task proposti per l'edizione 2022 sono stati:

2.1.1 Task 1: Carry My Luggage

Questo task prevede che il robot rilevi una borsa, la afferri e segua un operatore umano fino alla sua automobile per poi riconsegnargli la borsa.

2.1.2 Task 2: Find My Mates

Questo task richiede di rilevare un ospite, muoversi di fronte ad egli, poi recarsi di fronte ad un altro operatore umano e comunicare a quest'ultimo la posizione precisa dell'ospite. Inoltre, deve comunicare il nome dell'ospite ed una sua descrizione.

2.1.3 Task 3: Receptionist

Questo task vede il robot che intraprende un dialogo con un ospite, chiedendogli nome e drink preferito. In seguito, il robot dovrà comunicare le infor-

mazioni ottenute ad un operatore umano. Infine, il robot dovrà accompagnare l'ospite verso un posto a sedere vuoto.

2.2 Scelta del task

La scelta è ricaduta sull'ultimo task, quello del robot receptionist. La prima proposta era più appropriata per un robot che fosse dotato di un braccio meccanico ed il robot in dotazione non ne disponeva; la seconda consisteva quasi soltanto nel riconoscere gli ospiti; il terzo task invece mi è sembrato molto più interessante e formativo, comprendendo l'utilizzo sia di funzioni di speech recognition, di mapping e navigation e di computer vision (in particolare face recognition).

Capitolo 3

MARRtino

MARRtino è una piattaforma robotica a trasmissione differenziale a basso costo basata su ROS, disponibile in molte forme. È stato progettato per essere facile da costruire e facile da programmare, ma allo stesso tempo utilizza software professionali basati su ROS. È quindi adatto per implementare e sperimentare molti compiti tipici della Robotica e dell'Intelligenza Artificiale, come la navigazione intelligente, l'interazione vocale uomo-robot, l'analisi delle immagini e molto altro. MARRtino viene utilizzato con successo in una varietà di attività educative, che vanno dai bambini in età prescolare al PhD e PostDoc in Intelligenza Artificiale e Robotica.

3.1 Componenti hardware

Figura 3.1. Esemplare di robot utilizzato

Il robot utilizzato per lo svolgimento di questo tirocinio è stato gentilmente fornito dal DIAG e da Robotics 3D. Come si può notare dalla Figura 3.1, questo particolare esemplare è composto da:

- quattro ruote (di cui due motrici) per muoversi;
- una videocamera ASTRA per la cattura di immagini dal mondo esterno;
- un sistema di altoparlanti stereo per poter "parlare" con utenti esterni attraverso il Text-To-Speech (TTS).

Per quanto riguarda la parte di STT (Speech-To-Text), ovvero ciò che permette ad un utente esterno di parlare con il robot, si può notare che l'esemplare in questione non ha incorporato alcun "microfono". Questo perché la comunicazione "da umano a robot" avviene attraverso l'apposita applicazione per smartphone Android. Essa si avvale dell'oramai molto affidabile riconoscimento vocale di Google per sintetizzare e convertire in testo l'input vocale dell'utente, per poi essere inviato al robot per l'elaborazione.

3.2 Componenti software

Ciò che caratterizza MARRtino è proprio la semplicità con cui è possibile interagire con il nostro robot, e vengono messe a disposizione diverse opzioni:

- senza installazione (script Python via web browser web o VNC);
- macchina virtuale MARRtino (con VirtualBox);
- qualsiasi distribuzione Linux, fornita di Docker, Docker-Compose e marrtinoapps.

Durante i primi giorni ho utilizzato la macchina virtuale MARRtino, essendo lo strumento consigliato per avere un'esperienza già pre-configurata. Durante la mia attività di tirocinio, però, sono stati effettuati numerosi aggiornamenti volti a containerizzare il più possibile gli strumenti messi a disposizione. Ciò mi ha permesso di migrare, dopo aver adeguatamente compreso come utilizzare tali strumenti, verso la terza opzione. Nello specifico, ho utilizzato una distribuzione Ubuntu tramite il Sottosistema Windows per Linux (WSL), che fornisce prestazioni nettamente migliori rispetto alla più classica virtualizzazione con VirtualBox.

3.2.1 marrtinoapps

Il cuore di MARRtino sono le marrtinoapps. Esse mettono a disposizione dei container Docker che lavorano tra loro per permetterci di interagire con il nostro robot:

Teleop

Componente che ci permette di telecomandare il robot con l'ausilio di un joystick o della tastiera. Essa ci permette inoltre di scegliere se avvalerci o meno dell'evitamento degli ostacoli (obstacle avoidance).

Camera

Questa componente fa uso della sola videocamera per recuperare immagini dal mondo reale. Le funzioni di questo modulo permettono non solo di controllarla, ma danno anche la possibilità di lavorare su object recognition e face recognition. Purtroppo quest'ultima parte è poco sviluppata a causa della grande potenza di calcolo richiesta per eseguire queste operazioni.

Laser

Componente deputata alla gestione del sensore laser, necessario per l'obstacle avoidance e per il calcolo della distanza/profondità.

Audio

Server audio che si occupa del dialogo tra robot e umano.

Stage

Nel caso non si disponga di un robot fisico, è possibile avvalersi della simulazione 2D.

Marker

Rilevazione di AprilTags.

Navigation

Componente che gestisce lo spostamento nello spazio e l'obstacle avoidance. Lavora a stretto giro con la componente di mapping.

Mapping

Componente che ci permette di mappare l'ambiente, ad esempio la nostra casa. In questo modo possiamo ordinare al robot di andare in un determinato punto specificando le sue coordinate. Il robot effettua la mappatura in combinazione con la componente di navigation.

3.3 Interazione software-hardware

Il robot, all'accensione, espone un server accessibile collegandosi al suo access point Wi-Fi. Una volta connessi, possiamo avviare manualmente i singoli componenti o configurare un più conveniente script di autostart, per avviare le componenti che necessitiamo utilizzare in modo automatizzato. Inoltre, collegandoci via SSH all'indirizzo statico 10.3.1.1 (utente "marrtino") possiamo accedere a tutto ciò che è presente sul NUC del robot, scaricare gli aggiornamenti più recenti o apportare modifiche di qualsiasi tipo. Una volta avviato l'ambiente MARRtino, possiamo finalmente dedicarci all'attività di programmazione del robot.

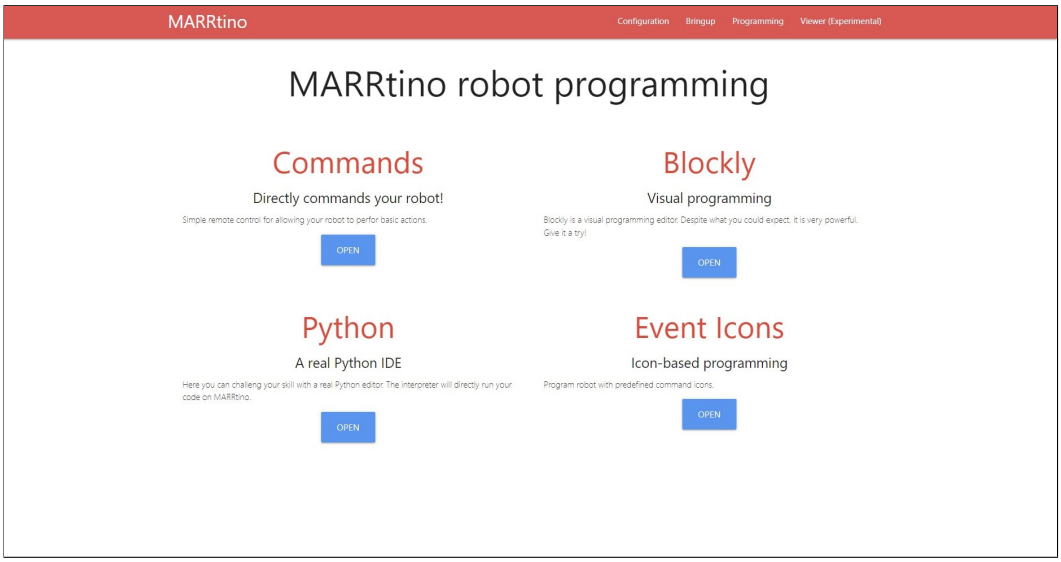

Figura 3.2. Interfaccia web MARRtino, sezione Programming

Come si può notare dalla Figura 3.2, per programmare il robot è possibile scegliere tra tre modalità:

3.3.1 Programmazione ad icone

Programmazione del robot con delle icone cliccabili. E' una modalità molto limitata, prevede infatti solo comandi basilari di movimento (avanti, indietro, destra e sinistra) e di riproduzione di suoni.

3.3.2 Programmazione Blockly

Programmazione del robot con dei blocchi di codice che rappresentano singole funzioni. Rappresentati come pezzi di puzzle, possono essere trascinati ed uniti tra loro grazie ad un'interfaccia grafica molto user-friendly. In questo modo si permette di programmare il robot anche a persone che non possiedono particolari skill di coding.

3.3.3 Programmazione Python

Programmazione del robot con Python, per mezzo di semplici funzioni. Ad esempio, la funzione forward(x) fa compiere x passi in avanti al robot; la funzione say(text, language) fa pronunciare al robot una frase in una lingua a nostra scelta; la funzione goto(x,y,th) ordina al robot di spostarsi nelle coordinate specificate. E' possibile programmare in Python sia eseguendo il codice nell'interfaccia web che all'interno del robot tramite l'ambiente SSH citato precedentemente.

Capitolo 4

Robot receptionist

In questo capitolo verrà trattato il task nella sua interezza, affrontando i suoi sottotask ed i problemi che presentano, spiegando poi alcuni dei metodi utilizzati per risolverli. Dunque, non verrà trattata l'implementazione pratica del task (che invece vedremo nel Capitolo 5), ma verranno elaborati in linea teorica i temi affrontati. Il task scelto per questo tirocinio è il task numero 3 della RoboCup@Home Education Challenge 2022. Esso presenta i seguenti sottotask:

- presentare il primo ospite a John
  - comunicare il nome del primo ospite;
  - comunicare il drink preferito del primo ospite;
  - guardare il primo ospite mentre lo si sta introducendo.
- offrire al primo ospite un posto a sedere
  - rilevare un posto vuoto;
  - guardare il posto vuoto mentre lo si offre.

I sottotask non rispetteranno questa suddivisione, ma verranno raggruppati per tema e problemi riscontrati, in modo da accorpare azioni simili ed evidenziare gli ostacoli affrontati nel lavoro.

4.1 Comunicazione con l'ospite e problema di face detection and recognition

La prima cosa che viene richiesta dal task è parlare con l'ospite. Infatti, per poterlo presentare all'operatore umano è necessario prima conoscerlo, il che implica l'averci parlato in precedenza. Tuttavia, prima ancora di interagire con l'ospite bisogna riconoscerlo (cioè sapere chi è un ospite e che caratteristiche ha). In altre parole, il robot deve poterlo distinguere dal resto dell'ambiente. Qui si presentano due possibili problemi a seconda del livello di raffinatezza che deve avere il task. Se si vuole che il robot riconosca semplicemente gli ospiti, ignorando le loro differenze, bisognerà affrontare un problema di face detection. Se invece si desidera che il robot sia in grado di distinguere i diversi ospiti, ci si dovrà confrontare con un problema di face recognition. Tratteremo questi due problemi in sottosezioni separate poiché nonostante sembrino simili nella teoria, nella pratica non lo sono poi così tanto.[1].

4.1.1 Face detection

Questo primo problema tratta il riconoscimento delle facce, ovvero la capacità di saper distinguere quello che è un viso da quello che non lo è. Non importa dunque chi sia il soggetto che si sta osservando, ma solo che abbia una faccia. Come categorizzato da Yang, Kriegman, e Ahuja [2] gli approcci possibili sono tre:

Feature-based

Questa tecnica si basa sul cercare di trovare tratti distintivi di una faccia (come gli occhi, la bocca ed il naso) e verificare che questi siano in posizioni plausibili.

Template-based

Questo approccio permette il riconoscimento di una faccia anche in diverse posizioni e con varie espressioni. Richiede una buona inizializzazione con un viso il più possibile vicino alla realtà. Non fornisce però un risultato rapido.

[1] Maggiori informazioni su questo genere di problemi si trovano nel libro Computer vision: algorithms and applications [1].

Appearence-based

Questa soluzione consiste nello scansionare piccoli frammenti rettangolari dell'immagine sovrapposti tra loro alla ricerca di buoni candidati ad essere una faccia. Per raffinare la ricerca, a questi possono essere poi applicati a cascata altri algoritmi di face detection. Per ovviare al problema del cambio di scala, l'immagine viene convertita in una piramide di sotto-ottave e i suoi livelli vengono scansionati separatamente.

4.1.2 Face recognition

Quando si parla di face recognition si intende il problema di distinguere un viso dall'altro, ovvero essere capaci, date due facce, di dire se appartengono o no alla stessa persona. Poter utilizzare questa funzionalità è utile sia per l'identificazione che per la validazione. Cioè, dato un volto, si può determinare se si conosce quella persona e, data un'identità, si può verificare se corrisponde a chi si ha davanti. Questo problema, per quanto diverso dal precedente, vi è legato. Infatti, per poter riconoscere una faccia in un'immagine è necessario sapere se è presente e in che punto si trova. Le soluzioni sono costituite di due parti che hanno funzioni diverse: la prima si occupa della raccolta dei dati dall'immagine di un viso, la seconda usa questi ultimi per fare dei confronti con altre immagini che gli vengono fornite. Gli approcci usati per risolvere questo problema sono cambiati nel tempo.

Approcci datati

Sfruttano le distanze tra le caratteristiche salienti in un viso, ovvero misurano gli spazi tra le parti della faccia come bocca, naso e occhi. Queste distanze dovrebbero essere sufficientemente adeguate per distinguere diverse persone. Purtroppo però non sono sempre così precise e sono state sostituite dove possibile da metodi più sofisticati. Ulteriori informazioni su questo tipo di approcci sono fornite dagli studi di Fischler e Elschlager [3] e di Yuille [4].

Approcci nuovi

Si basano sul comparare immagini in livelli di grigio proiettate in sottospazi di dimensioni inferiori, chiamate eigenfaces e, congiuntamente, modellare

variazioni di forma e aspetto usando active appearance models. Le eigenfaces si basano sull'osservazione, fatta per la prima volta da Kirby e Sirovich nel 1990 [5], che un'immagine arbitraria contenente una faccia possa essere compressa e ricostruita partendo da un'immagine principale e aggiungendo un piccolo numero di immagini riconosciute e scalate. Queste ultime possono essere derivate da un insieme di immagini di allenamento usando principal component analysis (anche conosciuta come Karhunen–Loeve transform [5]). Uno dei maggiori vantaggi dell'uso delle eigenfaces è che riducono le comparazioni da fare in un immagine per riconoscere una faccia [6] [7]. La riconoscibilità di un volto, come mostrato dal lavoro di Rowland e Perrett (1995) [8], dipende tanto dalla forma quanto dal colore o dalla texture; i due ricercatori sono riusciti tramite l'analisi di queste variabili ad associarvi caratteristiche personali come il sesso e l'età.

4.2 Comunicazione con l'ospite e problema di human-robot interaction

Il fulcro del task si trova proprio in questa parte, il che rende la Human-Robot interaction il problema principale da risolvere. Per poter comunicare con gli umani il robot deve essere in grado di fare tre cose:

- Ascoltare cosa è stato detto;
- Capire cosa è stato detto;
- Rispondere vocalmente.

Ognuna di queste parti è fondamentale e affronta tematiche diverse che verranno trattate in breve di seguito[2].

Ascoltare cosa è stato detto

Questa parte riguarda lo speech recognition, ovvero la capacità, dato un suono (una voce), di tradurlo in stringhe di testo, in modo che queste si possano analizzare per capire cosa è stato detto. Più precisamente si cerca di mappare

[2]Questi problemi sono meglio trattati nel libro Deep Learning di Ian Goodfellow, Yoshua Bengio ed Aaron Courville [9].

segnali acustici, tradizionalmente prodotti dividendo l'audio in frammenti della durata di 20ms, alla corrispondente sequenza di parole pronunciata dall'interlocutore. Gran parte dei sistemi di speech recognition preprocessano l'input usando funzionalità specifiche disegnate a mano, alcuni invece utilizzano il deep learning con le sue funzioni di apprendimento da input naturale. Lo speech recognition è stato uno dei primi campi in cui sono state adoperate le reti neurali, tuttavia esse non hanno avuto grande successo fino a tempi relativamente recenti. Il robot, per far fronte a questo problema, utilizza l'automatic speech recognition (ASR), una funzione che cerca di creare la più probabile sequenza linguistica dato il segnale in input.

Capire cosa è stato detto

Questo è uno dei problemi più grandi per un computer, ovvero quello di comprendere il significato di una frase o di un discorso. A differenza dei linguaggi di programmazione, il linguaggio naturale non è facilmente rappresentabile da un computer, per questo esiste il Natural Language Processing. Si tratta di una branca dell'informatica che si occupa di creare sistemi in grado di comprendere il linguaggio naturale. Riuscire a capire cosa l'interlocutore sta dicendo è fondamentale per poter selezionare una risposta adeguata e passare all'ultima parte della comunicazione.

Rispondere vocalmente

Quest'ultima fase della comunicazione tratta un problema che è forse il più semplice dei tre, ovvero il text to speech. Si vuole, come dice il nome, passare da un testo alla voce in modo da permettere alla persona con cui si sta interagendo di recepire la risposta. La sintesi vocale può essere realizzata concatenando registrazioni memorizzate in un database. I vari sistemi differiscono nelle dimensioni dei campioni vocali memorizzati: un sistema che memorizza singoli fonemi o fonemi doppi permette di ottenere il numero massimo di combinazioni a discapito della chiarezza complessiva, mentre in altri, concepiti per un impiego specifico, si ricorre alla registrazione di parole intere o di intere frasi per ottenere un risultato di qualità elevata. In alternativa, un sintetizzatore può incorporare un modello dei tratti vocali e di altre caratteristiche umane per creare una voce completamente di sintesi [10].

4.3 Spostamenti e problema di SLAM

Per poter individuare la sedia e portarci l'ospite bisogna affrontare alcuni dei principali problemi della programmazione di robot mobili. Sto parlando di navigation e mapping, queste due funzioni sono fondamentali per permettere al robot di essere autonomo negli spostamenti, senza di esse non avrebbe la consapevolezza dell'ambiente circostante e ciò renderebbe molto più difficile farlo muovere verso una determinata posizione e fargli evitare gli ostacoli. Quest'ultimo problema può comunque essere attenuato sfruttando funzioni di real time obstacle avoidance, ovvero adoperando il sensore laser del robot per rilevare gli ostacoli ogni volta che esso si sposta, e deviando il suo percorso in modo da evitarli. Un altro problema è la difficoltà nello stabilire con certezza dove si trovi il robot e come sia fatta la mappa a causa del noise presente nei dati ricavati dai sensori. Questo errore dipende appunto da quanto siano precise le rilevazioni fatte dal robot, ma anche se l'errore fosse molto piccolo nei lunghi percorsi sarebbe destinato ad accumularsi e crescere fino a impedire una navigazione corretta. I sensori che hanno questo problema non sono solo quelli per il rilevamento di ostacoli, come il sensore laser, ma anche quelli che stimano i movimenti del robot. Questi ultimi possono basarsi o sui giri delle ruote (MARRtino utilizza questo modello) oppure sulla velocità del robot e il tempo per il quale si muove. Questo problema è detto SLAM (Simultaneous Localization And Mapping) e per risolverlo molti algoritmi usano una stima probabilistica basata su diversi fattori: i rilevamenti dei sensori, le posizioni del robot stimate in precedenza e le posizioni stimate dei punti di riferimento. Per potersi orientare infatti anche il robot, come gli esseri umani, ha bisogno di punti di riferimento, che possono essere riconosciuti visivamente oppure individuati con il sensore laser, in quest'ultimo caso non si può avere la certezza di star osservando lo stesso punto. Prendendo i problemi di navigation e mapping singolarmente non si avrebbe particolare difficoltà a risolverli: se ad esempio si avesse una mappa già fatta, anche qualora lo spostamento del robot non fosse preciso, la sua posizione stimata potrebbe essere corretta verificando la distanza da determinati punti di riferimento. Tuttavia, questi due problemi dipendono l'uno dall'altro, il che rende ancora più difficile dare una stima accurata della posizione del robot e della locazione dei punti di riferimento. Gli algoritmi per risolvere SLAM procedono dunque per passi: prediction step

e correction step. Nel primo si sfruttano le informazioni sull'ambiente che già si possiedono e tramite il comando di spostamento che viene eseguito, si predice la posizione del robot. Nel passo successivo si acquisiscono nuovi dati dai sensori e tramite questi si cerca di correggere la posizione sia del robot che dei punti di riferimento. Esistono vari approcci per risolvere SLAM: Kalman filter, Extended Kalman filter, Information filter, Particle filter etc. [11].

4.4 Individuazione delle sedie e problema di object recognition

Una parte del task richiede di indicare all'ospite una sedia vuota, si presenta quindi il problema dell'object recognition. In particolare si cerca di distinguere una sedia vuota da una occupata. Si tratta quindi nello specifico di instance recognition, ovvero si cerca di riconoscere un oggetto ben noto, magari visto da diverse angolature e con diversi sfondi. Questa differisce dal riconoscimento generale di oggetti (category-level object recognition) il quale si concentra sull'individuazione di entità appartenenti a diverse classi e risulta per questo più difficile. Nel tempo sono stati sviluppati vari metodi per l'instance recognition, alcuni usano estrarre linee, contorni o superfici 3D dalle immagini per poi confrontarle a modelli 3D di oggetti specifici, altri si concentrano sull'acquisire immagini da un grande insieme di punti di vista e illuminazioni e rappresentarle usando l'eigenspace decomposition [12]. Gli approcci più recenti tendono invece ad usare le viewpoint-invariant 2D features. Dopo aver estratto le "informative sparse 2D features" dall'immagine da confrontare e da quelle nel database, queste sono comparate a coppie e, una volta trovato un numero sufficiente di aspetti in comune, si procede con una verifica cercando una trasformazione geometrica che le allinei.
Alcuni dei metodi più usati per l'object recognition sono basati su reti neurali, queste devono essere allenate con un adeguato set di immagini opportunamente etichettate in modo da imparare a riconoscere determinati oggetti. Ad esempio una rete neurale che deve riconoscere sedie vuote e sedie occupate deve essere addestrata su un insieme di immagini contenenti queste due classi. La qualità delle immagini e la loro quantità saranno fondamentali per generare un buon riconoscitore, questo perché la rete deve estrarre caratteristiche dalle immagini

di addestramento e più queste ultime saranno vicine all'oggetto che si va ad identificare, più il sistema sarà preciso. La quantità invece è necessaria per studiare gli aspetti simili e per avere una buona varianza che permetta di individuare un oggetto anche se non è estremamente vicino alle immagini di allenamento. Un aspetto importante, specifico del task, è invece il bilanciamento del database fornito per l'addestramento, c'è infatti bisogno che la differenza tra il numero di sedie vuote e occupate non sia troppo grande, se così fosse, durante l'allenamento, la rete potrebbe trovarsi a valutare alcuni batch contenenti solo immagini di un tipo e ciò potrebbe generare un riconoscitore poco affidabile. Per evitare ciò viene usata la k-fold cross-validation [13], la quale permette di valutare il funzionamento del riconoscitore fornendogli diversi insiemi di training data e validation data. In questo modo si può usare quello che si è dimostrato essere il migliore.

Capitolo 5

Implementazione del task

5.1 Ricognizione dell'ambiente esterno

Prima di iniziare con la scrittura del codice per eseguire il task scelto, è stato necessario mappare l'ambiente in cui il robot si trovava per far sì che esso potesse ricordare i luoghi attorno a lui e raggiungerli su comando.

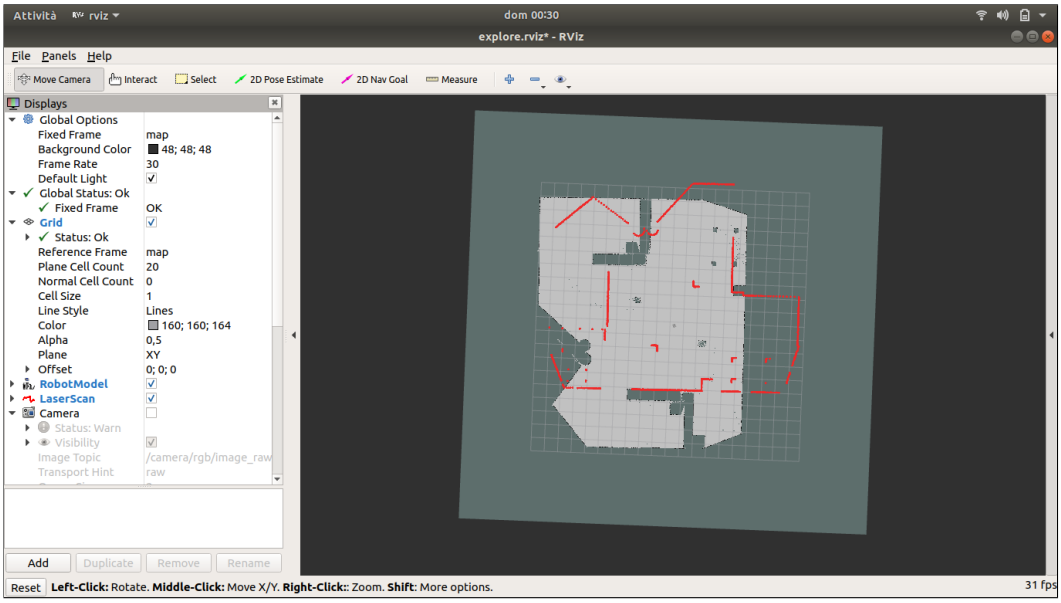

Figura 5.1. Screenshot di Rviz, software per la visualizzazione della mappa.

Come è possibile vedere dalla Figura 5.1, il software Rviz ci permette di seguire in tempo reale la mappatura che il robot sta eseguendo, vedendo la mappa formarsi man mano. Dopo aver effettuato la mappatura della mia

casa, ho definito delle funzioni Python che permettessero di raggiungere delle coordinate specifiche. Ad esempio, quando si invoca la funzione goToSpecificTarget("seat0"), al robot viene ordinato di posizionarsi di fronte alla prima sedia, per poter poi proseguire con il resto delle istruzioni.

```
...
def goToSpecificTarget(target):
    if target == "start":
        gotoTarget(0, 0, frame='map')
    elif target == "helper":
        gotoTarget(0.1, 0.08, frame='map')
    elif target == "seat0":
        gotoTarget(1.1, 0.16, frame='map')
    elif target == "seat1":
        gotoTarget(1.1, 0.24, frame='map')
    elif target == "seat2":
        gotoTarget(1.1, 0.32, frame='map')
...
```

5.2 Rilevazione del primo ospite

Come prima cosa è stato necessario far sì che il robot rilevasse un ospite da poter interrogare. Per portare a termine ciò è stato necessario implementare all'inizio del programma un ciclo deputato a catturare un'immagine e passarla alla funzione detectMultiScale della libreria OpenCV. Tale funzione rileva oggetti di diverse dimensioni nell'immagine di input, e per il nostro scopo è stato dunque necessario fornirle anche una Haar Cascade di facce frontali. Il ciclo summenzionato viene interrotto quando viene rilevato un viso.

Figura 5.2. Accoglienza di un ospite da parte del robot.

```
from time import sleep
import cv2

begin()

class Person:
    def __init__(self, face, name, gender, pronouns, drink):
        self.face = face
        self.name = name
        self.gender = gender
        self.pronouns = pronouns
```

```
        self.drink = drink

yes = ['yes', 'Yes', 'YES', 'Yeah', 'yeah', 'Yep', 'yep', 'YEP',
    'Correct', 'correct', 'Right', 'right']
no = ['no', 'No', 'Nope', 'nope', 'Wrong', 'wrong']
maleGender = ['male', 'mail', 'man', 'men', 'boy', 'guy', 'Male',
    'Mail', 'Man', 'Men', 'Boy', 'Guy']
femaleGender = ['female', 'woman', 'women', 'girl', 'gal',
    'Female', 'Woman', 'Women', 'Girl', 'Gal']

# Si preleva il modello Haar Cascade per facce frontali
faceCascade = findCascadeModel()

# Funzione per prelevare l'ultima immagine catturata, in formato JPG
def getImageJPG():
    getImage()
    MARRTINO_APPS_HOME = os.getenv('MARRTINO_APPS_HOME')
    assert MARRTINO_APPS_HOME is not None
    return MARRTINO_APPS_HOME + "/www/viewer/img/lastimage.jpg"

...

faces = [] # Deve avere grandezza pari ad 1, ovvero una faccia alla
    volta
facesJPG = [] # Stessa condizione dell'array "faces"
guests = [] # Array con unico scopo di logging (memorizzare i
    profili degli ospiti)
helperName = "John" # Nome dell'helper (nel nostro caso "John")

while (faces == []):
    # Legge l'immagine catturata
    image = cv2.imread(getImageJPG(), 0)
    # Rileva le facce nell'immagine
    faces = faceCascade.detectMultiScale(
        image,
        scaleFactor=1.1,
        minNeighbors=5,
```

```
        minSize=(30, 30)
    )
    # Handler nel caso di rilevazione di due o piu' facce (vedere
        Paragrafo 5.2.1)
    if faces > 1:
        say("I see more than one face, please one of you step back.
            Will check again in 2 seconds.")
        faces = []
        sleep(2)
        continue
    # Ritaglia la faccia e la memorizza in un array
    for (x, y, w, h) in faces:
        facesJPG.append(image[y:y+h, x:x+w])
    ...
```

5.2.1 Rilevazione di più di un ospite nello stesso momento

Cosa succede nel caso venisse rilevato più di un ospite nella stessa immagine? In tal caso, sono state valutate due soluzioni:

- il robot deve chiedere agli ospiti di posizionarsi in modo che solo un ospite alla volta venga rilevato. Il robot tornerà al punto di partenza ogni volta che termina di gestire un ospite;
- il robot memorizza tutti gli ospiti rilevati e li gestisce uno alla volta.

Nonostante siano molto simili, la seconda opzione è lievemente migliore. Il problema presentatosi, però, è che, essendo l'immagine elaborata da OpenCV come una matrice, lo storing dei volti nell'array è ordinato in modo da partire dal volto più in alto a sinistra fino al volto più in basso a destra. Certamente non è fattibile chiedere agli ospiti di disporsi in un certo ordine. Ho ritenuto dunque necessario optare per la prima opzione. La differenza, come già detto in precedenza, è marginale, dato che in ambedue le soluzioni il robot gestisce un ospite alla volta. Il codice Python che gestisce questa casistica è stato già mostrato nello snippet di codice presente al Paragrafo 5.2.

5.3 Comunicazione con l'ospite

Si giunge ora ad una parte prettamente incentrata sullo speech. Tramite le funzioni say() e asr(), il robot:

- chiede all'ospite il suo nome ed in seguito lo ripete chiedendo se sia corretto
  - se non è corretto, viene chiesto di ripetere il nome (e viene nuovamente chiesta conferma, così via fino a quando l'ospite non ci da una risposta affermativa).

Lo stesso procedimento si applica per le domande sul genere e sul drink preferito.

5.3.1 Linguaggio gender-neutral

Il robot porrà all'ospite anche una quarta domanda:

- "Con quale genere ti identifichi di più?"

Ho ritenuto questa domanda necessaria per un corretto e rispettoso utilizzo dei pronomi nei dialoghi seguenti.

```
...
# Mostra la faccia dell'ospite (solo su robot provvisti di
    display)
display(facesJPG[0])
# Chiede il nome dell'ospite
name = ""
say("Hello, what is your name?", language="en")
nameIsCorrect = False
while (nameIsCorrect == False):
    name = asr()
    say("Is " + name + " correct?", language="en")
    if (asr() in yes):
        nameIsCorrect = True
        name = name.capitalize()
    else:
```

```
        say("Oh, sorry, I must've heard wrong! Please repeat.",
            language="en")
# Chiede il sesso dell'ospite
gender = ""
pronouns = ""
say("How do you identify yourself?", language='en')
genderIsCorrect = False
while (genderIsCorrect == False):
    gender = asr()
    if gender in maleGender:
        say('You identify as a male, correct?', language='en')
        if (asr() in yes):
            genderIsCorrect = True
            gender = "male"
            pronouns = "he/him/his"
        else:
            say("Oh, sorry, I must've heard wrong! Please
                repeat.", language="en")
    elif gender in femaleGender:
        say('You identify as a female, correct?', language='en')
        if (asr() in yes):
            genderIsCorrect = True
            gender = "female"
            pronouns = "she/her/her"
        else:
            say("Oh, sorry, I must've heard wrong! Please
                repeat.", language="en")
    else:
        say('You identify as non-binary, correct?', language='en')
        if (asr() in yes):
            genderIsCorrect = True
            gender = "non-binary"
            pronouns = "they/them/their"
        else:
            say("Oh, sorry, I must've heard wrong! Please
                repeat.", language="en")
# Chiede il drink preferito dell'ospite
```

```
drink = ""
say("Last question: what's your favorite drink?", language='en')
drinkIsCorrect = False
while (drinkIsCorrect == False):
    drink = asr()
    say("Is " + drink + " correct?", language="en")
    if (asr() in yes):
        drinkIsCorrect = True
        drink = drink.capitalize()
    else:
        say("Oh, sorry, I must've heard wrong! Please repeat.",
            language="en")
# Crea un oggetto persona attraverso la classe dedicata
person = Person(facesJPG[0], name, gender, pronouns, drink)
# Memorizza l'oggetto persona in un array per questioni di
    logging
guests.append(person)
...
```

5.4 Raggiungimento di John

Dopo aver memorizzato i dati dell'ospite, il robot chiede all'ospite di seguirlo per raggiungere John, grazie alla funzione integrata gotoTarget(x, y, map). Il robot è dunque al corrente dei luoghi che lo circondano grazie alla mappatura, e si ipotizza che John sia un collega umano posizionato sempre in un determinato posto. Data la mancanza di face recognition causa limitazioni hardware del robot, il robot chiederà all'ospite di rimanere sempre alla sua sinistra, in modo da sapere già dove puntare quando necessario.

5.4.1 John non è presente

Cosa succede nel caso John non fosse presente in quel momento? In tal caso, arrivato a destinazione il robot:

- cattura un'immagine e controlla se in essa è presente almeno un volto

– se nessun volto è presente, dice ad alta voce di aspettare che John torni. Effettuerà un nuovo tentativo dopo 5 secondi.

```
...
# Il robot va da John
say("Okay, " + name + ", please follow me. And please always
    stay on my left.")
goToSpecificTarget("helper")
# Controlla se John e' li'
helperIsThere = False
while (helperIsThere == False):
    # Read the image
    image = cv2.imread(getImageJPG(), 0)
    # Detect faces in the image
    temp = faces
    faces = faceCascade.detectMultiScale(
        image,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30)
    )
    if faces != []:
        helperIsThere = True
    else:
        say("I can't find the helper, please wait. Will check
            again in 5 seconds.")
        sleep(5)
    faces = temp
...
```

5.5 Guardare l'ospite durante la sua introduzione

Uno dei sottotask richiede che il robot guardi l'ospite mentre egli viene presentato a John. Dunque, il robot, dopo aver rilevato e salutato John, dovrà

nuovamente puntare l'ospite.

Figura 5.3. Introduzione dell'ospite.

```
...
# Introduzione dell'ospite a John
say("Hi " + helperName + "! This is our new guest!",
    language='en')
# Trova l'ospite (per guardarlo durante l'introduzione)
left(1)
say(pronouns.split("/")[2] + " name is " + name + " and " +
    pronouns.split("/")[2] + " favorite drink is " + drink +
    ".", language='en')
```

```
# Guarda John
right(1)
say("I'll now bring " + pronouns.split("/")
    [1] + " to an empty seat!", language='en')
# Guarda l'ospite
left(1)
...
```

5.6 Rilevamento di un posto libero

Una volta completata l'introduzione dell'ospite a John, il robot dovrà accompagnare l'ospite a sedersi. Per rilevare un posto effettivamente libero, ho fatto uso di un AprilTag. Esso è definito come un sistema fiduciale visivo, utile per un'ampia varietà di attività, tra cui realtà aumentata, robotica e calibrazione della fotocamera. I target possono essere creati da una normale stampante e il software di rilevamento AprilTag calcola la posizione 3D precisa, l'orientamento e l'identità dei tag relativi alla fotocamera.

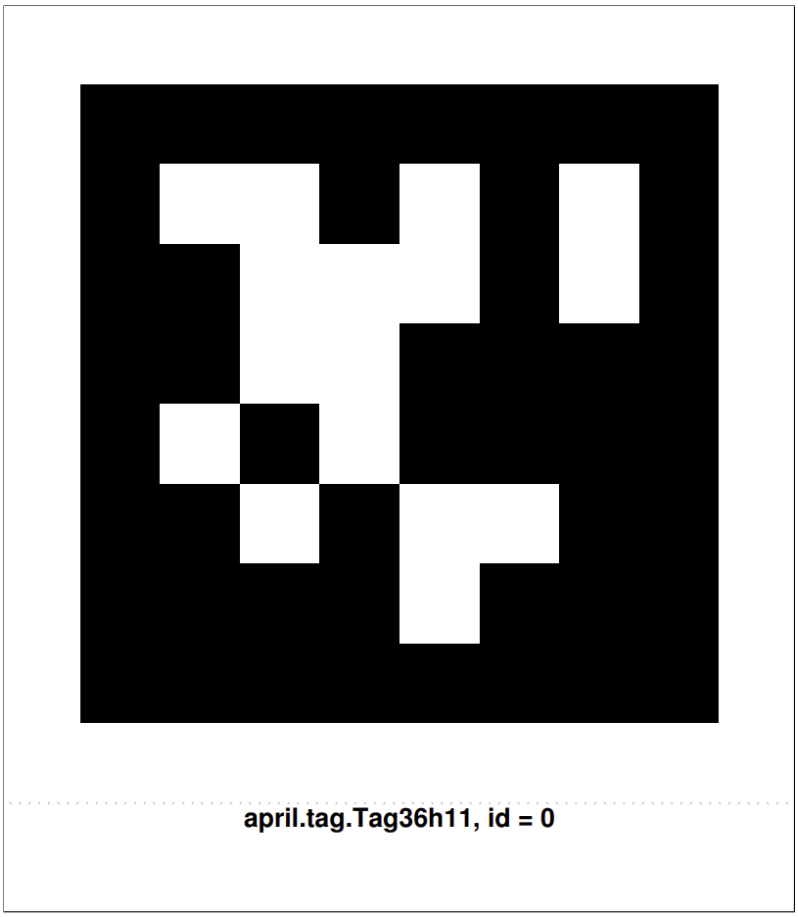

Figura 5.4. Uno dei tag riconosciuti da MARRtino.

Nel mio caso ho utilizzato una stanza con tre sedie ed una poltrona, sullo schienale di ogni quale ho posto un foglio con un AprilTag. In questo modo, se una persona occupa un determinato posto a sedere, l'AprilTag sullo schienale è coperto (e dunque non in vista). Il robot chiederà nuovamente all'ospite di seguirlo e si recherà nella sala dei posti a sedere, nello specifico davanti alla prima sedia. A questo punto, il robot:

- cattura un'immagine e controlla se in essa è presente un AprilTag
  - se è presente un AprilTag, propone all'ospite di sedersi lì;
  - se non è presente un AprilTag, si sposta alla posizione della seconda sedia, e così via. Nel caso anche la terza ed ultima sedia risulti occupata, dirà all'ospite che purtroppo non ci sono posti liberi e che potrà attendere che qualcuno si alzi. Infine, il robot saluterà e

tornerà alla posizione di partenza (per potersi occupare del prossimo ospite).

Figura 5.5. Rilevazione posto libero tramite un AprilTag.

```
...
# Vai alla stanza dei posti a sedere
say("Please follow me to the seats room.", language='en')
goToSpecificTarget("seat0")
say("There you go, " + name +
    "! Let's see if there is an empty seat for you!",
        language='en')
seatPointing = "seat0"
```

```
    emptySeat = False
    while emptySeat == False:
        if tagTrigger() == True:
            emptySeat = True
            say("You can sit there! I'll go, you can find me at the
                reception.", language='en')
            goToSpecificTarget("start")
        if (seatPointing == "seat0"):
            goToSpecificTarget("seat1")
            seatPointing = "seat1"
        elif (seatPointing == "seat1"):
            goToSpecificTarget("seat2")
            seatPointing = "seat2"
        # se tutti i posti a sedere sono stati controllati, il robot
            comunichera' all'ospite che non c'e' nessun posto vuoto
            in quel momento
        elif (seatPointing == "seat2"):
            say("There is no empty seat at the moment. You can wait
                here until another guest stands up. I'll go, you can
                find me at the reception.", language='en')
            goToSpecificTarget("start")
            break

end()
```

Capitolo 6

Considerazioni post-sviluppo

Nel corso dello sviluppo della soluzione di questo task ho dovuto effettuare delle scelte importanti per poter soddisfare le diverse necessità del progetto, basandomi su diversi criteri, tra cui la realizzabilità di alcune implementazioni. In questo capitolo esporrò le mie riflessioni su come sia possibile non solo migliorare la soluzione al task, ma anche apportare degli enhancement per un eventuale utilizzo in un setting pratico di vita reale.

6.1 Object recognition

Per trovare una sedia libera, ho utilizzato degli AprilTags attaccati sullo schienale di ogni sedia, dato che tale tag sarebbe stato visibile dal robot solo in caso di sedia non occupata. L'altra opzione sarebbe stata l'object recognition. MARRtino dispone di funzioni deputate a questo scopo, supportate da librerie come TensorFlow. La creazione di un riconoscitore automatico di sedie, però, non è stata ritenuta fattibile dato che:

- Avrebbe richiesto un egual numero di immagini di sedie vuote e di dover allenare una rete neurale;
- Libreria presente sul robot era molto datata (TensorFlow 1) ed aggiornarla avrebbe richiesto dei cambiamenti profondi alle marrtinoapps (le quali vengono mantenute dal Prof. Iocchi);
- Problemi di performance. L'hardware del robot non è idoneo a svolgere funzioni di object recognition. Per questi scopi vengono di norma utilizzati

computer con caratteristiche di altissimo livello. Sarebbe pienamente possibile implementare questa soluzione connettendo il robot ad un server in grado di elaborare tali richieste da remoto, in modo da poter svolgere queste funzioni in maniera efficiente e soprattutto in tempistiche ragionevoli.

6.2 Face recognition

Quanto appena detto per l'object recognition si applica anche alla face recognition: sull'hardware del robot essa richiede molto tempo per soddisfare una singola richiesta, il che non è affatto idoneo ad un utilizzo nella vita reale. Proviamo a modificare la soluzione per soddisfare il sottotask che chiedeva di puntare l'ospite durante la sua introduzione. Invece di assumere che l'ospite è sempre alla sua sinistra, il robot, che sta guardando John, deve girare su sé stesso in senso antiorario fino ad individuare il viso dell'ospite. Considerando il caso peggiore, ovvero che l'ospite sia sulla sinistra del robot, quest'ultimo dovrà compiere ben quattro face comparison. Il tutto senza tener conto delle tempistiche necessarie alla cattura dell'immagine, altra operazione che richiede anch'essa qualche secondo di attesa. Nonostante non abbia avuto a disposizione l'hardware adatto, ho comunque scritto una versione alternativa del programma che fa uso della face recognition tramite l'omonima API per Python [14].

6.3 Natural Language Processing

Potrebbe anche essere migliorata la parte di conversazione uomo-macchina, utilizzando tecniche di Natural Language Processing (NLP) per elaborare in modo più accurato le frasi ricevute. La soluzione implementata, invece, si limita a confrontare le parole ricevute con degli array di parole predefiniti, che comprendono sia le parole precise che delle parole fuori contesto ma assonanti. Ad esempio:

```
maleGender = ['male', 'mail', 'man', 'men', 'boy', 'guy']
```

In questo array notiamo che è presente la parola "mail". Questo perchè è capitato che il riconoscimento vocale da smartphone comprendesse tale parola invece di "male" a causa di una pronuncia errata della lingua inglese.

6.4 Speech recognition dal robot

Il modello di robot utilizzato è sprovvisto di un microfono, e potremmo pensare sia l'unica motivazione per cui è necessario l'utilizzo di uno smartphone Android per dialogare con il robot. In realtà, sarebbe teoricamente possibile implementare un software Python con il quale è possibile dialogare con le Speech API di Google, le stesse di cui usufruisce Android per elaborare lo Speech-To-Text. Il tutto avverrebbe grazie alle librerie Python SpeechRecognition (per eseguire il riconoscimento vocale con supporto a diversi motori ed API, online ed offline) e PyAudio (per la gestione dei canali audio, supportata anche da aziende leader come Dolby). Ovviamente, questo richiederebbe comunque la presenza di un microfono sul robot e di un account Google configurato alla fruizione delle Google Speech API.

6.5 Dialogare con un database invece che con John

Uno dei sottotask richiedeva che il robot riferisse le informazioni ad un operatore umano, John, prima di proseguire con la ricerca di un posto a sedere. Si potrebbe abbattere notevolmente il tempo necessario per gestire un ospite facendo inviare al robot i dati dell'ospite ad un server database. In questo modo il robot non avrebbe bisogno di riferire le informazioni a John e passerebbe subito alla ricerca di un posto a sedere, rendendo più efficiente il sistema per l'utilizzo in un setting di vita reale. Ancora meglio, anche i posti a sedere si potrebbero gestire tramite database, in modo che il robot verifichi subito se esiste un posto libero sul momento e gestirebbe l'ospite di conseguenza.

Capitolo 7

Conclusioni

Durante questo lavoro ho dovuto in primis impegnarmi a comprendere il funzionamento dei robot ed in particolare di MARRtino. Ho reperito tali informazioni dal sito ufficiale e, dopo aver installato ed inizializzato l'ambiente di sviluppo, ho esplorato il suo codice per comprenderne le funzioni di base. Successivamente ho affrontato il task della RoboCup@Home Education Challenge, il quale mi ha portato a ragionare su vari temi come face detection e face recognition, tag detection ed object detection, localizzazione e mapping simultanei. Alcuni di questi campi si sono rivelati molto semplici da studiare ed attuare, grazie specialmente alle funzioni di MARRtino già pronte all'uso; altri task si sono rivelati più complicati, come ad esempio la face detection e l'object recognition. Per migliorare la soluzione di questo task ci sono molti metodi, ad esempio utilizzare l'object recognition per individuare le sedie: ciò sarebbe pienamente possibile in remoto, connettendo il robot ad un server in grado di elaborare tali richieste in condizioni ottimali. Lo stesso si applica alla face recognition: nonostante sia stata implementata, essa richiede diversi secondi per soddisfare ogni richiesta, il che non è affatto idoneo ad un utilizzo nella vita reale. Potrebbe anche essere migliorata la parte di conversazione uomo-macchina, utilizzando tecniche di Natural Language Processing (NLP) per elaborare in modo più accurato le frasi ricevute. Da questo lavoro ho imparato molto su come funziona un robot, sono venuto a conoscenza di varie tecniche utilizzate per il machine learning ed ho imparato a gestire un lavoro che consiste nello sviluppo di software che necessita di interagire in tempo reale con l'hardware. In alcuni momenti mi sono trovato in difficoltà, ma ciò mi ha spronato a cercare di capire come superare gli ostacoli. Ho anche avuto

modo di conoscere due laureandi (Alessandro Monteleone e Ludovico Comito) che stavano anch'essi svolgendo un task della competizione come tirocinio. Confrontarmi con loro mi è servito molto e li ringrazio di cuore. Ringrazio inoltre il Professor Iocchi che, nonostante i suoi numerosi impegni, si è sempre rivelato disponibile e pronto a chiarire qualsiasi mio dubbio con precisione e celerità. Questo progetto di tirocinio mi è servito a capire che le sfide più interessanti sono quelle più complicate, ma se vengono svolte con passione e dedizione regalano davvero tante soddisfazioni.

Bibliografia

[1] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.

[2] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[3] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions on computers*, vol. 100, no. 1, pp. 67–92, 1973.

[4] A. L. Yuille, "Deformable templates for face recognition," *Journal of cognitive neuroscience*, vol. 3, no. 1, pp. 59–70, 1991.

[5] M. Kirby and L. Sirovich, "Application of the karhunen-loeve procedure for the characterization of human faces," *IEEE Transactions on Pattern analysis and Machine intelligence*, vol. 12, no. 1, pp. 103–108, 1990.

[6] Y. Yu *et al.*, "Face recognition with eigenfaces," in *Proceedings of 1994 IEEE International Conference on Industrial Technology-ICIT'94*, pp. 434–438, IEEE, 1994.

[7] A. Lemieux and M. Parizeau, "Experiments on eigenfaces robustness," in *2002 International Conference on Pattern Recognition*, vol. 1, pp. 421–424, IEEE, 2002.

[8] D. A. Rowland and D. I. Perrett, "Manipulating facial appearance through shape and color," *IEEE computer graphics and applications*, vol. 15, no. 5, pp. 70–76, 1995.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[10] A. Acero, "An overview of text-to-speech synthesis," in *2000 IEEE Workshop on Speech Coding. Proceedings. Meeting the Challenges of the New Millennium (Cat. No. 00EX421)*, p. 1, IEEE, 2000.

[11] T. Sebastian, B. Wolfram, and F. Dieter, "Probabilistic robotics," 2005.

[12] H. Murase and S. K. Nayar, "Learning and recognition of 3d objects from appearance," in *[1993] Proceedings IEEE Workshop on Qualitative Vision*, pp. 39–50, IEEE, 1993.

[13] Y. Bengio and Y. Grandvalet, "No unbiased estimator of the variance of k-fold cross-validation," *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[14] A. Geitgey, "Face recognition." https://github.com/ageitgey/face_recognition, 2017.