Progetto Finale di Cognitive Robotics: Gruppo 26

Giovanni Ammendola Edoardo Maffucci Vincenzo Petrone
Salvatore Scala

21 gennaio 2021

g.ammendola4@studenti.unisa.it, e.maffucci1@studenti.unisa.it,
v.petrone6@studenti.unisa.it, s.scala6@studenti.unisa.it

Indice

1 Introduzione **1**
1.1 Repository GitHub . . . 1
1.1.1 Utilizzo . . . 1

2 Descrizione dei nodi **2**
2.1 Head Node . . . 2
2.2 Camera Node . . . 2
2.3 Detector Node . . . 3
2.3.1 Scelta del detector . . . 3
2.4 Speaker Node . . . 4

3 Meccanismo di comunicazione tra nodi **5**
3.1 Messaggi . . . 5
3.1.1 Messaggio DetectionWithScore . . . 5
3.2 Servizi . . . 6
3.2.1 Servizio TakePicture . . . 6
3.2.1.1 Richiesta . . . 6
3.2.1.2 Risposta . . . 6
3.2.1.3 Client . . . 6
3.2.1.4 Server . . . 6
3.2.2 Servizio DetectImage . . . 6
3.2.2.1 Richiesta . . . 6
3.2.2.2 Risposta . . . 6
3.2.2.3 Client . . . 6
3.2.2.4 Server . . . 6
3.2.3 Servizio SayDetections . . . 7
3.2.3.1 Richiesta . . . 7
3.2.3.2 Risposta . . . 7
3.2.3.3 Client . . . 7
3.2.3.4 Server . . . 7

1 Introduzione

Contenuto del presente documento è la descrizione dell'implementazione di un software ROS che permette al robot sociale Pepper di riconoscere oggetti nella scena di fronte a sé e descrivere la scena stessa, elencando i differenti oggetti che ha visto e specificando in quale direzione sono stati rilevati.

1.1 Repository GitHub

La repository GitHub su cui è possibile visualizzare il codice e la documentazione è raggiungibile a questo link: `https://github.com/CR2020-team/cogrob_final-project`.

1.1.1 Utilizzo

Per testare correttamente il software, si seguano le istruzioni specificate sulla repository: `https://github.com/CR2020-team/cogrob_final-project#usage`.

2 Descrizione dei nodi

I 4 nodi sviluppati sono implementati nella cartella `https://github.com/CR2020-team/cogrob_final-project/tree/master/pepper_pkg/src`.
Utilizzano metodi implementati nella libreria `naoqi`, i cui differenti proxy permettono di astrarre i compiti eseguiti dai vari nodi.

2.1 Head Node

Il nodo chiamato `Head Node` si occupa di muovere la testa di Pepper in tre direzioni: destra, centro, sinistra.
Utilizza i proxy `ALRobotPosture` e `ALMotion`.
Per raggiungere le posizioni desiderate, imposta gli angoli di giunto `HeadPitch` ed `HeadYaw`: il primo è pari a 15°, per permettere a Pepper di guardare in basso verso la scrivania; il secondo varia, per permettere a Pepper di guardare nelle tre direzioni.
Per guardare a destra, l'angolo `HeadYaw` è impostato a -56.3°.
Per guardare dritto, l'angolo `HeadYaw` è impostato a 0°.
Per guardare a sinistra, l'angolo `HeadYaw` è impostato a 56.3°.
Il valore 56.3° è scelto perché equivale al FOV orizzontale della camera frontale superiore di Pepper, come riportano le specifiche tecniche: `http://doc.aldebaran.com/2-5/family/pepper_technical/video_2D_pep_v18a.html#video-2d-pepper-v18a`. Questa scelta fa in modo che le immagini acquisite non siano sovrapposte, dunque in esse non saranno presenti oggetti duplicati.
Per far sì che Pepper guardi sempre negli stessi punti, viene riportata nella posizione `StandInit`, in cui si trova dritta in piedi, guardando di fronte a sé, con le braccia lungo il corpo.
Ogni volta che Pepper raggiunge una delle tre direzioni, chiama il servizio `TakePicture` per chiedere al `Camera Node` di acquisire un'immagine in quella direzione.

2.2 Camera Node

Il nodo chiamato `Camera Node` si occupa di acquisire un'immagine dalla camera frontale superiore di Pepper.
Utilizza il proxy `ALVideoDevice`.
Dopo aver registrato la camera in fase di inizializzazione, crea un service server per il servizio `TakePicture`. Ogni volta che esso viene invocato, acquisisce un'immagine e la trasforma in una `ROS Image`, che diventa a sua volta parte della richiesta al servizio `DetectImage`, il quale fa in modo che il `Detector Node` rilevi oggetti nell'immagine acquisita.

2.3 Detector Node

Il nodo chiamato `Detector Node` si occupa di rilevare oggetti nelle immagini acquisite.
Dopo aver caricato il detector in fase di inizializzazione, crea un service server per il servizio `DetectImage`. Ogni volta che esso viene invocato, inserisce l'immagine richiesta in una coda, i cui elementi verranno dati in input al detector. La label e il relativo *confidence score* di ogni oggetto rilevato vengono inseriti in un messaggio `DetectionWithScore`.
Gli oggetti che possono essere rilevati sono quelli appartenenti al dataset `COCO`, le cui label sono disponibili su GitHub: `https://gist.github.com/xhlulu/f7735970704b97fd0b72203628c1cc77`.
Tuttavia, alcune classi sono state "scartate", in quanto il compito chiedeva che Pepper riconoscesse oggetti che tipicamente possono essere trovati in un appartamento. Per cui, le classi da 1 a 26 non sono state prese in considerazione. È possibile recuperare le classi rimanenti e le relative label nella repository GitHub: `https://github.com/CR2020-team/cogrob_final-project/blob/master/pepper_pkg/src/detector_node/classmap.py`.
Per ogni immagine ricevuta, viene inoltrata una richiesta al servizio `SayDetections`, che contiene una lista di `DetectionWithScore` per ogni oggetto rilevato nell'immagine acquisita e la direzione richiesta. Tali messaggi verranno poi letti dallo `Speaker Node`.

2.3.1 Scelta del detector

La scelta del particolare detector da utilizzare è stata guidata da due obiettivi: il primo è limitare il tempo di caricamento del modello e di inferenza, il secondo è aumentare il più possibile le performance. Dunque, la scelta è ricaduta su quel modello che, in fase di test, ha raggiunto un buon compromesso tra i due obiettivi.
A tal fine, sono stati considerati diversi modelli di EfficientDet che, stando alla tabella riportata da `tensorflow`, sono tra le reti che offrono il maggior mAP per un tempo di esecuzione ragionevole e una dimensione del modello ridotta. In particolare, sono stati testati i modelli D0, D1, D2, D3 di EfficientDet, di cui sono state riportate le performance in termini di tempo nella tabella seguente. Infine, il modello scelto è stato **EfficientDetD1**, in quanto è stato registrato un miglioramento nell'efficacia in tempi ragionevoli.

Detectors		
Model	**Tempo di caricamento (s)**	**Tempo medio di inferenza (s)**
EfficientDetD0	66	4
EfficientDetD1	**87**	**4,67**
EfficientDetD2	104	7
EfficientDetD3	121	12,3

2.4 Speaker Node

Il nodo chiamato `Speaker Node` si occupa di fare in modo che Pepper elenchi ad alta voce gli oggetti presenti nella scena, specificandone anche il numero e la direzione in cui gli stessi sono stati rilevati.
Utilizza il proxy `ALAnimatedSpeech` per permettere a Pepper di gesticolare in modo naturale mentre parla; a tal fine, il suo linguaggio del corpo viene impostato in modalità `contextual`.
Il nodo è un service server per il servizio `SayDetections`: per ogni richiesta ricevuta, esegue i seguenti passi:

- Crea un dizionario le cui chiavi sono le label degli oggetti unici presenti nel campo `detections` della richiesta e i cui valori sono il numero di volte in cui la relativa label si ripete.
  - Ad esempio, se `detections` contiene `['bottle', 'umbrella', 'potted plant', 'bottle']`, allora il dizionario conterrà {`'bottle':2, 'umbrella':1, 'potted plant':1`}.
  - In sostanza, il dizionario conterrà quante volte un oggetto è stato rilevato in ciascuna immagine.
- Trasforma ogni coppia chiave-valore del dizionario in una descrizione testuale dell'oggetto, la quale contiene la label (eventualmente al plurale) e il relativo numero (o articolo indeterminativo se il numero è pari a 1). Ad esempio:
  - la coppia `'bottle':2` viene convertita nella stringa `2 bottles`.
  - la coppia `'umbrella':1` viene convertita nella stringa `an umbrella`.
  - la coppia `'potted plant':1` viene convertita nella stringa `a potted plant`.
- Tutte le stringhe così ottenute vengono unite in un'unica stringa del tipo `2 bottles, an unmbrella and a potted plant`.
- A questa stringa viene aggiunta la direzione contenuta nel campo `direction` del messaggio, ottenendo una nuova stringa del tipo `2 bottles, an unmbrella and a potted plant on the right`.

Una volta che è stata ricevuta una richiesta per ciascuna delle 3 direzioni, le stringhe vengono unite in un'unica stringa, ottenendo un testo del tipo `I can see: a laptop on the right, a tv, a potted plant, a mouse, 2 bottles and a keyboard in front of me, nothing on the left`.
Il testo così ottenuto viene processato dal proxy: in questo modo Pepper può finalmente elencare a voce tutti gli oggetti che ha visto nella scena.

3 Meccanismo di comunicazione tra nodi

I nodi comunicano attraverso messaggi e servizi, implementati nella cartella `https://github.com/CR2020-team/cogrob_final-project/tree/master/pepper_msgs`.
Si è valutato l'utilizzo di altri meccanismi di comunicazione quali Publisher/Subscriber e ROS actions: il primo tipo di meccanismo non è stato scelto perché tipicamente usato per comunicazioni broadcast o multicast, non utili nel nostro caso; le ROS actions, d'altro canto, prevedono comunicazioni end-to-end ma, a differenza dei ROS services, sono non bloccanti. Dunque, per aver un maggior controllo sul flusso delle operazioni e sui dati scambiati tra i vari nodi, si è deciso di utilizzare il meccanismo dei ROS services.
Nonostante il tipo di comunicazione scelto preveda della chiamate a funzioni bloccanti, l'esecuzione del sistema è veloce perché ciascun nodo esegue delle operazioni non costose computazionalmente (ad esempio, aggiunta in coda ad una lista, acquisizione di un'immagine). Inoltre, l'impiego dei ROS services è servito per bloccare l'esecuzione delle funzionalità dei nodi al verificarsi di alcuni errori a runtime, dovuti ad esempio ad un malfunzionamento dei componenti del robot (ad esempio, la camera).

3.1 Messaggi

I messaggi sono definiti nella cartella `https://github.com/CR2020-team/cogrob_final-project/tree/master/pepper_msgs/msg`.

3.1.1 Messaggio DetectionWithScore

Il messaggio `DetectionWithScore` rappresenta un oggetto rilevato e il relativo *confidence score*. È composto da due campi:

- `clabel`: una stringa che rappresenta la label dell'oggetto rilevato
- `score`: un numero reale nel range (0,1) che rappresenta il *confidence score* della rilevazione

Non esiste un topic dedicato a questo tipo di messaggio, in quanto è unicamente utilizzato nel campo `detections` della richiesta del servizio `SayDetections`

3.2 Servizi

I servizi sono definiti nella cartella `https://github.com/CR2020-team/cogrob_final-project/tree/master/pepper_msgs/srv`.

3.2.1 Servizio TakePicture

Il servizio `TakePicture` è utilizzato per fare in modo che Pepper acquisisca un'immagine in una data direzione.

3.2.1.1 Richiesta

La richiesta consiste in un unico campo `direction`, un intero in {-1; 0; 1} che codifica la direzione alla quale acquisire l'immagine: -1 indica destra, 0 indica il centro, 1 indica sinistra.

3.2.1.2 Risposta

La risposta consiste in un unico campo `success`, un booleano vero quando l'operazione avviene con successo, falso altrimenti.

3.2.1.3 Client

L'unico client del servizio `TakePicture` è l'`Head Node`, che fa una richiesta ogni volta che la testa giunge in una posizione desiderata.

3.2.1.4 Server

Il server del servizio `TakePicture` è il `Camera Node` che, ricevuta la direzione alla quale acquisire l'immagine, inoltra la direzione e l'immagine acquisita al servizio `DetectImage`.

3.2.2 Servizio DetectImage

Il servizio `DetectImage` è utilizzato per fare in modo che il detector rilevi oggetti in una data immagine.

3.2.2.1 Richiesta

La richiesta consiste in due campi:

- `image`: l'immagine su cui il detector farà inferenza, rilevando gli oggetti in essa presenti
- `direction`: un intero in {-1; 0; 1} che codifica la direzione alla quale l'immagine è stata acquisita; -1 indica destra, 0 indica il centro, 1 indica sinistra

3.2.2.2 Risposta

La risposta consiste in un unico campo `success`, un booleano vero quando l'operazione avviene con successo, falso altrimenti.

3.2.2.3 Client

L'unico client del servizio `DetectImage` è il `Camera Node`, che fa una richiesta ogni volta che acquisisce un'immagine.

3.2.2.4 Server

Il server del servizio `DetectImage` è il `Detector Node` che, ricevuta l'immagine e la direzione alla quale essa è stata acquisita, rileva gli oggetti in essa presenti, li trasforma una lista di `DetectionWithScore` e crea una richiesta per il servizio `SayDetections` in cui il campo `detections` è la lista creata e il campo `direction` è la direzione ricevuta.

3.2.3 Servizio SayDetections

Il servizio `SayDetections` è utilizzato per fare in modo che una lista di oggetti rilevati venga convertita in una stringa.

3.2.3.1 Richiesta

La richiesta consiste in due campi:

- `detections`: una lista di `DetectionWithScore` che rappresenta gli oggetti rilevati
- `direction`: un intero in {-1; 0; 1} che codifica la direzione in cui gli oggetti in `detections` sono stati rilevati; -1 indica destra, 0 indica il centro, 1 indica sinistra

3.2.3.2 Risposta

La risposta consiste in un unico campo `success`, un booleano vero quando l'operazione avviene con successo, falso altrimenti.

3.2.3.3 Client

L'unico client del servizio `SayDetections` è il `Detector Node`, che fa una richiesta ogni volta che esegue inferenza su un'immagine.

3.2.3.4 Server

Il server del servizio `SayDetections` è lo `Speaker Node`, che legge le richieste e le converte in testo come descritto nel paragrafo dedicato.