Università degli Studi di Bergamo

SCUOLA DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica

Laboratorio di Robotica

Documentazione attività di laboratorio

Prof.
Davide Brugali

Candidati
Giulia Allievi
Matricola 1058231

Martina Fanton
Matricola 1059640

Anno Accademico 2022–2023

Indice

Introduzione

L'obiettivo del nostro progetto è quello di fornire le coordinate e l'orientamento della pinza ad un robot, al fine di potersi posizionare per afferrare un oggetto. Le coordinate e l'orientamento vengono calcolate da un'opportuna libreria (nel nostro caso dalla libreria *Grasp Pose Generator, GPG*), la quale necessita delle informazioni dell'oggetto inquadrato sotto forma di una nuvola di punti, la point cloud, che verrà fornita attraverso la *Stereo Camera ZED*. Il software del progetto sarà realizzato utilizzando il *Framework STAR*.

La *Stereo Camera ZED* è dotata di vari sensori, per esempio l'accelereometro e il giroscopio, ma nella nostra applicazione serviranno solo i dati delle immagini, in particolare solo quelli della point cloud. La fotocamera stereoscopica ha un range di profondità compreso fra 30 cm e 20 m (fonte: datasheet), perciò gli oggetti da inquadrare dovranno essere posizionati ad una distanza compresa in questo intervallo.

Di seguito, nel capitolo 2 verranno analizzate due diverse librerie che si possono utilizzare per ottenere i dati relativi alla posizione e all'orientamento della pinza del robot, mentre nel capitolo 3 verranno descritti i passi necessari per integrare, compilare ed eseguire il progetto. Infine, nel capitolo 4 verranno mostrati i risultati finali.

Librerie

Prima di costruire l'applicazione finale (in cui è stata integrata soltanto la libreria GPG), sono state analizzate due diverse librerie *stand-alone*, la *Grasp Pose Generator* (GPG), la cui documentazione è reperibile al seguente link, e la *Grasp Pose Detection* (GPD), la cui documentazione è invece disponibile al seguente link.

2.1 Grasp Pose Generator (GPG)

Questa libreria permette di individuare, data la point cloud di un oggetto contenuta in un file di tipo *.pcd*, un certo numero di possibili pinze con cui il robot può afferrare l'oggetto stesso. Questo numero di pinze restituite in output viene definito dall'utente nel file di configurazione della libreria.

Per poter eseguire la libreria, innanzitutto si deve scaricare lo zip della libreria da Github (link) estraendone il contenuto, rinominandolo come `grasp_candidates_generator` e spostandolo in una directory scelta dall'utente. Poi vanno scaricate tutte le librerie che la libreria GPG richiede per il suo funzionamento, ovvero *PCL* (necessaria per elaborare la point cloud), *Eigen* (necessaria per eseguire operazioni con vettori e matrici) e *VTK* (necessaria per la visualizzazione della point cloud), aprendo un nuovo terminale ed eseguendo in esso i comandi:

```
sudo apt install libpcl-dev
sudo apt install libeigen3-dev
sudo apt-get install vtk6
```

Ora si può procedere con l'esecuzione della libreria GPG. Per prima cosa si effettua la compilazione eseguendo nel precedente terminale i seguenti comandi:

```
cd <location_of_your_workspace>
cd grasp_candidates_generator
mkdir build && cd build
cmake ..
make
sudo make install
```

Successivamente va eseguito anche il prossimo comando per l'esecuzione effettiva della libreria:

```
./generate_candidates ../cfg/params.cfg ~/data/<point_cloud_name_file>.pdc
```

In questo modo si visualizza a video la point cloud dell'oggetto in uno spazio tridimensionale, però per visualizzare anche le pinze individuate dalla libreria è necessario cliccare nella tastiera sul tasto Q. Per chiudere anche questa nuova finestra va premuto nuovamente il tasto Q.

Mettere alla fine un'immagine con la visualizzazione delle prese e delle info date dal terminale

2.2 Grasp Pose Detection (GPD)

A differenza della libreria precedente, questa libreria restituisce in output soltanto le pinze individuate che hanno lo score maggiore rispetto a tutte quelle disponibili. Queste pinze vengono scelte in un numero definito sempre dall'utente nel file di configurazione della libreria.

Per poter eseguire la libreria, innanzitutto si deve scaricare lo zip della libreria da Github (link) estraendone il contenuto, rinominandolo come `gpd` e spostandolo in una directory scelta dall'utente. Poi vanno scaricate tutte le librerie che la libreria GPD richiede per il suo funzionamento, ovvero *PCL* (necessaria per elaborare la point cloud), *Eigen* (necessaria per eseguire operazioni con vettori e matrici), *VTK* (necessaria per la visualizzazione della point cloud) e *OpenCV* (necessaria per eseguire la detection delle pinze), aprendo un nuovo terminale ed eseguendo in esso i comandi:

```
sudo apt install libpcl-dev
sudo apt install libeigen3-dev
sudo apt-get install vtk6
sudo apt install libopencv-dev
```

Ora si può procedere con l'esecuzione della libreria GPD. Per prima cosa si effettua la compilazione eseguendo nel precedente terminale i seguenti comandi:

```
cd <location_of_your_workspace>
cd gpd
mkdir build && cd build
cmake ..
make -j
sudo make install
```

Successivamente va eseguito anche il prossimo comando per l'esecuzione effettiva della libreria:

```
./detect_grasps ../cfg/eigen_params.cfg ../tutorials/krylon.pdc
```

Questo comando viene eseguito per la point cloud contenuta nel file *krylon.pcd* fornito da esempio nella libreria, ma nel comando può essere inserito qualsiasi altro file di tipo *.pcd*.

In questo modo si visualizza a video la point cloud dell'oggetto in uno spazio tridimensionale, però per visualizzare anche le pinze individuate dalla libreria è necessario cliccare nella tastiera sul tasto Q. Le pinze visualizzate vengono distinte in base al loro colore secondo lo score che le caratterizza. Per chiudere anche questa nuova finestra va premuto nuovamente il tasto Q.

Mettere alla fine un'immagine con la visualizzazione delle prese e delle info date dal terminale

Integrazione

In questo capitolo, verranno descritti i passi seguiti per ottenere l'applicazione finale. Innanzitutto è importante inizializzare le variabili d'ambiente di STAR in un nuovo terminale. Per fare ciò, bisogna eseguire questi comandi nel terminale:

```
cd STAR
cd AutonomousRobots
. env-star.sh
```

A questo punto, si inserisce la password (che è `unibg`) ed il terminale è pronto per essere utilizzato.

3.1 Libraries

Per prima cosa, andiamo a copiare e compilare la libreria GPG descritta nella sezione precedente nella sottocartella `others` della cartella `Libraries` di STAR. Per fare ciò eseguiamo i seguenti comandi in un terminale, dopo aver raggiunto la cartella `others`:

```
cd <location_of_your_workspace>
cd grasp_candidates_generator
mkdir build && cd build
cmake ..
make
sudo make install
```

Per eseguire la libreria, si deve digitare il comando seguente all'interno della cartella `build`:

```
./generate_candidates ../cfg/params.cfg ~/gpg/tutorials/<point_cloud_name_file>.pcd
```

La libreria appena importata funziona fornendogli in ingresso un file con estensione `.pcd` che contiene le informazioni sui punti della point cloud, però nel nostro caso la point cloud non sarà salvata su un file, ma verrà fornita dalla fotocamera stereoscopica. Modifichiamo quindi la libreria per fare in modo che riceva in ingresso non un file, ma una struttura dati denominata `CloudCamera` che è un riferimento alla struttura dati `pcl::PointCloud<pcl::PointXYZRGBA>::Ptr`. Quest'ultima variabile è un vettore contenente dei punti nel formato XYZRGBA: ogni punto è caratterizzato da 4 campi, i primi tre (X, Y, e Z) sono di tipo double e contengono le informazioni relative alle coordinate di quel punto in metri,

invece l'ultimo campo, RGBA, è un intero senza segno a 32 bit in cui si codifica l'informazione sul colore. Ogni parametro del colore (R, G, B e A) è memorizzato in 8 bit e questo significa che ognuno di questi valori può variare da 0 a 255, perciò possono essere rappresentati fino a $2^{32} = 4.294.967.296$ colori. Nello specifico, gli 8 bit meno significativi contengono l'informazione relativa al colore blu (B), i bit da 9 a 16 contengono l'informazione del colore verde (G), i bit da 17 a 24 codificano il valore del rosso (R), infine i bit da 25 a 32 contengono l'informazione del parametro alfa (A), che indica la trasparenza del colore. Nella nostra applicazione non utilizzeremo quest'informazione perché la point cloud è formata solo da punti di colore nero, che viene codificato dal valore più basso, ovvero 0. Per ottenere la modifica appena descritta, cambiamo il parametro della funzione `CandidatesGenerator::generateGraspCandidates`, che non sarà più una stringa contenente il percorso del file, ma una varibalie di tipo `const CloudCamera&`.

La seconda modifica che apportiamo riguarda il tipo restituito dalla funzione `CandidatesGenerator::generateGraspCandidates`, perché non sarà più di tipo `std::vector<Grasp>` ma `void`. Il vettore delle prese viene aggiornato passando alla funzione l'indirizzo della variabile da aggiornare, perciò la funzione di partenza avrà un parametro aggiuntivo.

A seguito di queste due modifiche, la segnatura della funzione `CandidatesGenerator::generateGraspCandidates` è:

```
void CandidatesGenerator::generateGraspCandidates(const CloudCamera&, std::vector<Grasp>&)
```

e nel main modifichiamo la chiamata della funzione, creando anche una struttura dati di tipo `std::vector<Grasp>` per memorizzare le prese valide:

```
std::vector<Grasp> candidates;
candidates_generator.generateGraspCandidates(cloud_cam,candidates);
```

Dopo aver modificato la libreria, è necessario ricompilarla andando ad eseguire l'ultimo comando della sequenza di compilazione illustrata in precedenza, che è:

```
sudo make install
```

È estremamente importante ripetere questo passaggio ogni volta che si desidera apportare delle modifiche alla libreria.

3.2 Functionalities

Dopo aver importato e modificato la libreria GPG creiamo il plugin, che servirà per richiamare l'esecuzione della libreria all'interno di un'attività di STAR. Il plugin va creato nella sottocartella `plugin` della cartella `Functionalities` di STAR.

Tutto quello che serve per creare e far funzionare il nostro plugin si trova nella cartella `arm_grasp_gcg`. Questa cartella contiene a sua volta altre tre cartelle (`cfg`, `lib` e `src`) e un `makefile`. La cartella `lib` contiene un file con estensione `.so` che corrisponde alla libreria dinamica che crea la libreria GPG, la

cartella `cfg` contiene un file che riassume tutti i parametri configurabili della libreria GPG in formato XML, infine la cartella `src` contiene il file che permette di esportare in STAR la libreria GPG come plugin.

Quest'ultimo file, `PluginArmGraspGCG.cpp` è stato ottenuto copiando e modificando il file della libreria GPG che contiene il metodo `main` (ovvero `generate_candidates.cpp`). Per prima cosa, la funzione `main` è stata tolta e trasformata in una normale funzione, all'inizio del file si registrano tutti i parametri di configurazione che successivamente vengono inizializzati ed infine si esporta il plugin aggiungendo le seguenti istruzioni:

```
extern "C" BOOST_SYMBOL_EXPORT PluginArmGraspGCG arm_grasp_gcg;
PluginArmGraspGCG arm_grasp_gcg;
```

Successivamente, si crea il `makefile` che contiene le istruzioni per compilare questo plugin. Le prime istruzioni riguardano il nome della libreria e il file in cui viene richiamata:

```
LIB_NAME       = arm_grasp_gcg
LIB_SRC        = PluginArmGraspGCG.cpp
```

Successivamente, vanno specificati i nomi ed i percorsi delle librerie necessarie affinché il plugin funzioni correttamente. Le dipendenze sono quelle indicate nella sezione 2.1, perciò specifichiamo il percorso delle librerie *PCL*, *Eigen* e *VTK*.

Per quanto riguarda il file di configurazione (`arm_grasp_gcg.cfg`), i parametri di nostro interesse sono:

- `finger_width`: spessore delle dita della pinza utilizzata;
- `hand_outer_diameter`: diametro della pinza, calcolato come somma fra la massima apertura della pinza e due volte lo spessore delle dita;
- `hand_depth`: lunghezza delle dita della pinza;
- `hand_height`: spessore (inteso come profondità) della pinza;
- `voxelize`: variabile booleana che è `true` se si vuole che i punti della point cloud vengano voxelizati al fine di ridurne il numero, vale `false` altrimenti;
- `remove_outliers`: variabile booleana utilizzata per indicare se si vogliono tenere tutti i valori (in questo caso il suo valore è `false`) oppure se si voglio scartare quelli statisticamente lontani al fine di ridurre il rumore della point cloud (in quest'altro caso il valore della variabile è `true`);
- `num_samples`: numero di campioni, ovvero di prese, che si vogliono tenere in considerazione. Nel caso in cui si è interessati ad una sola presa il suo valore è 1, altrimenti è un intero maggiore di 1.

Infine, va creata un'interfaccia all'interno della cartella `Functionalities` in cui richiamiamo la funzione della libreria. Quest'interfaccia va creata nel percorso `Functionalities/interfaces/star/robotics/functionality` e contiene gli `include` necessari e una funzione virtual che richiama la libreria. La sua segnatura è la seguente:

```
virtual void generateGraspCandidates(pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud-
Prova, std::vector<Grasp>\& prese) = 0;}
```

Dopo aver costruito il plugin, è necessario compilarlo, eseguendo le seguenti istruzioni:

```
cd STAR/AutonomousRobots/Functionalities/plugin/ arm_grasp_gcg
make
```

Il plugin va ricompilato ogni volta che viene modificato.

3.3 Components

Abbiamo creato due diverse attività: `ObjectDetection` si occupa dell'utilizzo della libreria GPG e della pubblicazione di messaggi contenenti le informazioni sulla presa, mentre `ObjectVisualization` si limita a ricevere e a stampare sul terminale il messaggio ricevuto dalla prima attività.

attività sono ObjectDetection e ObjectVisualization, mentre componente è CameraZedServer

3.4 Messages

Per quanto riguarda i messaggi, per la nostra applicazione è risultato necessario utilizzare i messaggi di tipo `manipulation_msgs` disponibili nella cartella *Messages* di STAR e in particolare abbiamo dovuto introdurre nel file `manipulation_msgs.idl` una nuova tipologia di struttura dati denominata `GraspCandidates`. Questa struttura consente di contenere al suo interno le possibili pinze individuate dalla libreria GPG ed è composta a sua volta da un campo corrispondente a una sequenza di oggetti di tipo `GraspPose`. Questa è un'ulteriore struttura dati che abbiamo creato in modo da rappresentare la singola pinza ed è costituita da tre campi di tipo *Transform3D*, che a sua volta è una nuova struttura dati introdotta che serve per rappresentare la traslazione e la rotazione della pinza. Questi tre campi si distinguono in base al tipo di traslazione individuata:

- `coord_bottom`, che riguarda la traslazione rispetto alla posizione centrata sulla base della pinza del robot;
- `coord_top`, che riguarda la traslazione rispetto alla posizione centrata sulle estremità delle dita della pinza del robot;
- `coord_surface`, che riguarda la traslazione rispetto alla posizione centrata sulla superficie dell'oggetto.

Transform3D è formata da 7 campi di tipo double, di cui i primi tre permettono di rappresentare le traslazioni (x, y e z), mentre gli ultimi quattro rappresentano le rotazioni espresse sotto forma di quaternioni (x, y, z e w).

Una volta apportate le precedenti modifiche al file `manipulation_msgs.idl`, è necessario ricompilare i messaggi tramite i seguenti comandi in un nuovo terminale:

```
cd STAR/AutonomousRobots
. env-star.sh
cd Messages
. compile.sh
```

Questa compilazione va eseguita tutte le volte che si modificano i messaggi.

Risultati

Immagini con prese, eventualmente anche quelle con integrazione robot.