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: https://github.com/atenpas/gpg, e la *Grasp Pose Detection* (GPD), la cui documentazione è invece disponibile al seguente link: https://github.com/atenpas/gpd.

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. I parametri contenuti nel file di configurazione sono descritti più in dettaglio nella sezione 3.2.

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 e delle prese), 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>.pcd
```

In questo modo si visualizza a video un disegno in cui vengono visualizzati i vettori normali alla superficie inquadrata, per visualizzare la point cloud dell'oggetto in uno spazio tridimensionale con le pinze individuate dalla libreria è necessario cliccare il tasto Q sulla tastiera. Per chiudere anche questa nuova finestra va premuto nuovamente il tasto Q. Per entrambe le finestre, per cambiare lo zoom bisogna usare lo scroller del mouse, per muoversi sulla point cloud bisogna invece trascinare con il cursore. Nella figura 2.1 sono riportate le due immagini.

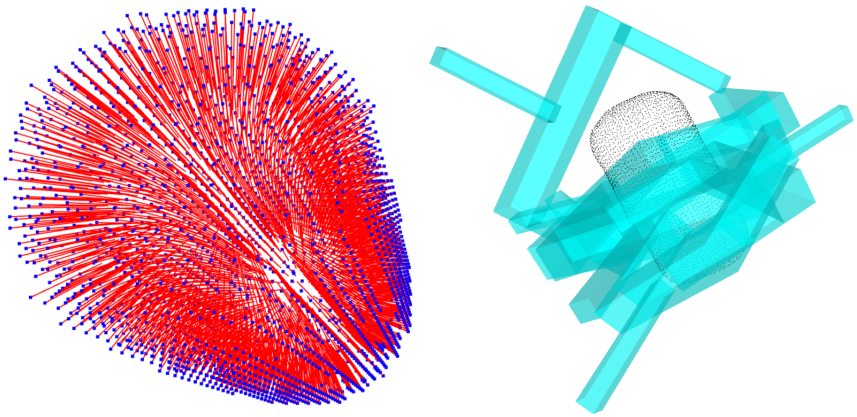

Figura 2.1: Vettori normali alla superficie (a sinistra) e point cloud con sei possibili prese (a destra).

2.2 Grasp Pose Detection (GPD)

La libreria *Grasp Pose Detection* è l'evoluzione della libreria *Grasp Pose Generator* appena descritta. A differenza della libreria precedente, in questa libreria è integrato anche un classificatore per riconoscere gli oggetti afferrabili. La libreria GPD restituisce in output le pinze individuate, analogamente alla libreria GPG, ma per ogni presa viene specificato anche il suo score, che nell'immagine viene reso utilizzando dei diversi colori (verde per le pinze con score maggiore, che sono quelle corrette; giallo e arancione per quelle di score intermedio, che sono le prese accettabili; rosso per quelle con score basso, che sono da considerarsi non accettabili). Anche per questa libreria il numero delle pinze è definito dall'utente nel file di configurazione.

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 delle prese) 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.pcd
```

Questo comando viene eseguito sulla point cloud contenuta nel file `krylon.pcd` fornito da esempio dalla libreria, ma nel comando può essere inserito qualsiasi altro file di tipo `.pcd`.

In questo modo si visualizzano a video i vettori normali alla superficie della point cloud in uno spazio tridimensionale, però per visualizzare anche le pinze individuate dalla libreria è necessario cliccare nella tastiera sul tasto Q. Per vedere il numero scelto di pinze migliori, che vengono distinte in base al loro colore secondo lo score che le caratterizza, è necessario premere di nuovo il tasto Q. Per chiudere anche questa nuova finestra va premuto nuovamente il tasto Q. Le immagini descritte sono riportate in figura 2.2, invece le informazioni sullo score mostrate a terminale si trovano in figura 2.3.

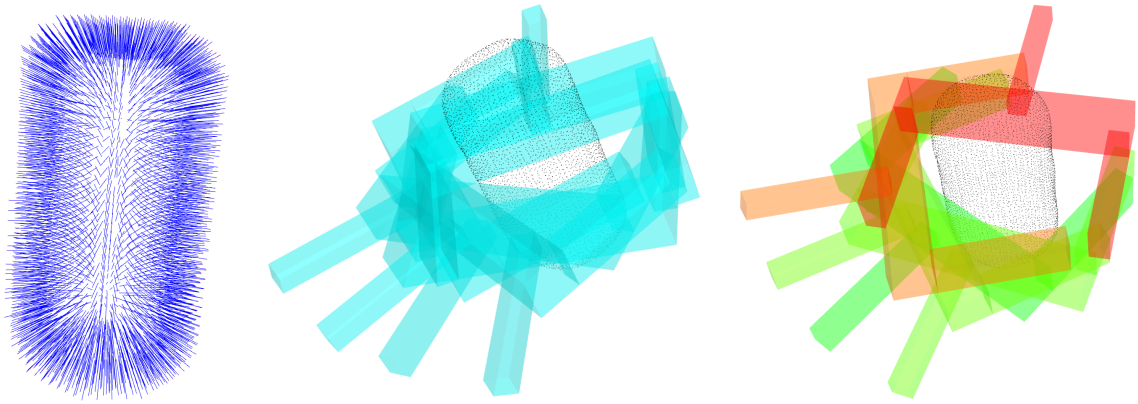

Figura 2.2: Vettori normali alla superficie (a sinistra), point cloud con le possibili prese (al centro) e point cloud con le cinque migliori prese (a destra).

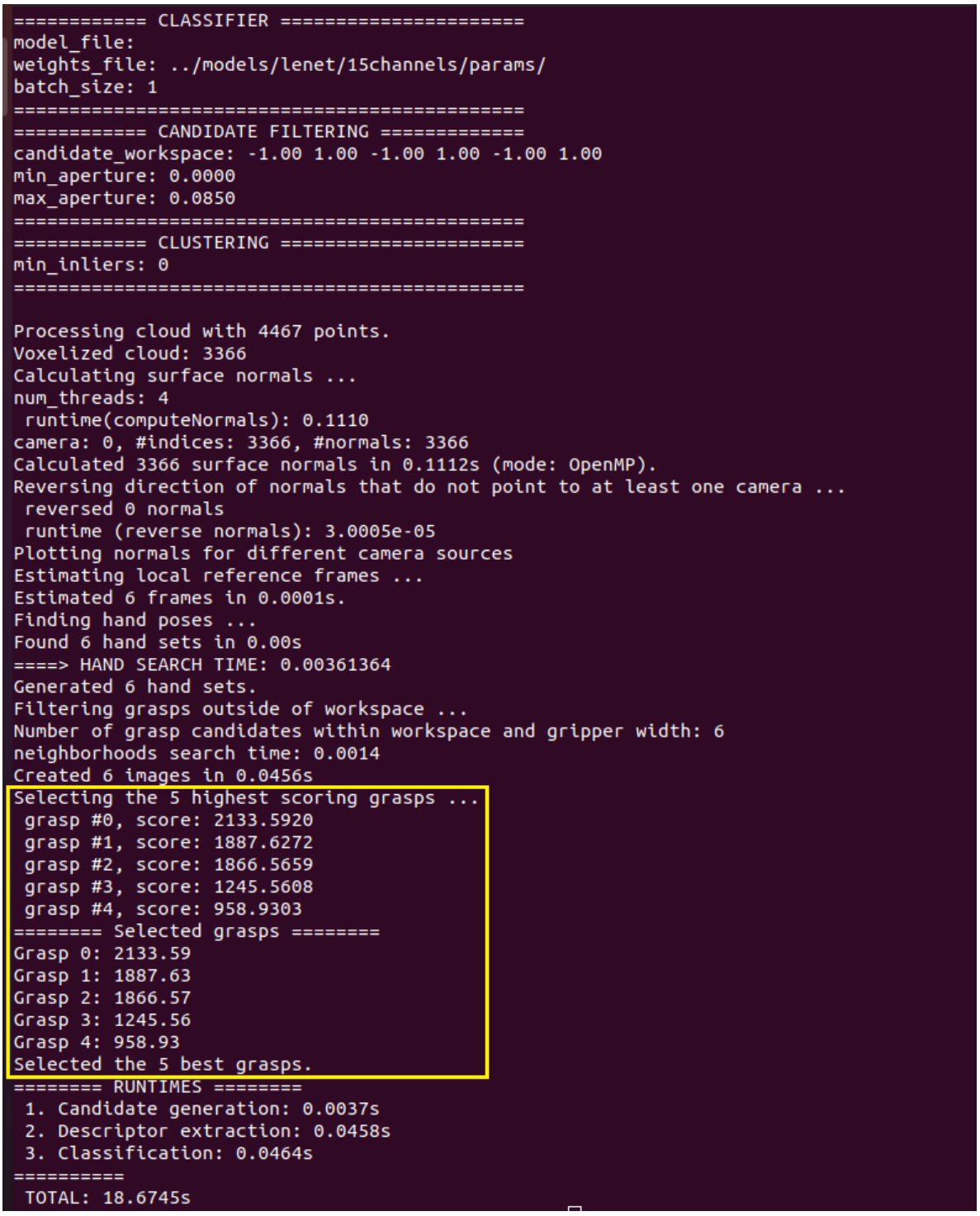

Figura 2.3: Informazioni mostrate a terminale, quelle relative allo score si trovano nel riquadro evidenziato.

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. L'inizializzazione va ripetuta ogni volta che si vuole utilizzare un nuovo terminale per eseguire delle azioni su STAR.

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 gpg
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
~/STAR/AutonomousRobots/Libraries/others/gpg/tutorials/krylon.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 (`generate_candidates.cpp`) 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 vogliono 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;
- `plotNormals` e `plotGrasps`: sono entrambe variabili booleane, sono `true` se si è interessati a visualizzare il relativo grafico, altrimenti sono `false`.

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

Per la nostra applicazione utilizzeremo il componente `CameraZedServer`, che si trova all'interno della cartella `Components` di STAR. In questo componente troviamo quattro diverse cartelle: in `bin` troviamo i file di log e gli script per compilare ed eseguire il componente; all'interno di `cfg` troviamo un file in formato XML che contiene le informazioni relative alle attività, ad esempio di quali topic sono publisher o subscriber ed il dominio sul quale si scambiano eventuali messaggi; infine `src` contiene i file `.cpp` e `.hpp` delle diverse attività del componente.
Le attività di nostro interesse sono elencate di seguito:

- `ImageGrabbing`: attività periodica che ha il compito di salvare le immagini provenienti dalla fotocamera stereoscopica in uno slot del CAB (Circular Asynchronous Buffer);
- `ImageVisualization`: è l'attività sporadica che consuma lo slot dell'attività precedente, il suo compito è quello di visualizzare tutte le immagini fornite dalla fotocamera;
- `ObjectDetection`: attività creata da noi, il suo compito è quello di richiamare il plugin `arm_grasp_gcg` per fornire le coordinate delle prese tramite un messaggio, verrà descritta dettagliatamente in seguito;
- `ObjectVisualization`: attività sporadica creata da noi, ha il compito di stampare sul terminale le coordinate delle prese che riceve tramite messaggio dal componente precedente.

Nel file di configurazione abbiamo aggiunto le due attività create, in particolare per l'attività `ObjectDetection` è stata inserita anche la chiamata al plugin ed è stato abilitato il log su disco, questo ci servirà durante la progettazione del codice a scopo di debug.

Prima di procedere, è necessario conoscere il sistema di riferimento adottato dalla telecamera, così da capire il significato, in termini di coordinate, dei dati che ci restituirà con la point cloud. La Stereo Camera ZED mette a disposizione dell'utente sei diversi sistemi di riferimento, che sono mostrati in figura 3.1. Noi abbiamo scelto il primo, perciò le informazioni relative alla distanza dell'oggetto saranno

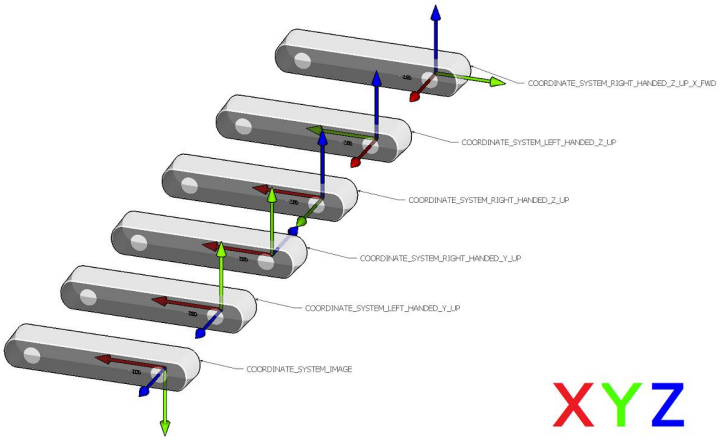

Figura 3.1: Sistemi di rifermento messi a disposizione dalla nostra fotocamera stereoscopica.

date dalla coordinata X.

Di seguito, analizziamo la struttura del file relativo all'attività `ObjectDetection`. Le prime due funzioni dell'attività sono il costruttore e il distruttore della classe, mentre la terza funzione è la callback relativa alla connessione. La callback relativa alle informazioni delle fotocamera, invece, è organizzata nel seguente modo:

- *Acquisizione delle immagini dallo slot del CAB*: nella nostra applicazione serve solo la point cloud, che viene memorizzata in una struttura dati di tipo `cv::Mat`;
- *Rilascio dello slot del CAB appena utilizzato*: in seguito al salvataggio della point cloud, possiamo rilasciare lo slot del CAB per renderlo nuovamente disponibile a contenere nuovi dati;
- *Impostazione delle soglie*: è **facoltativa**. Dal momento che questa libreria è sprovvista del classificatore, le soglie servono per individuare l'oggetto da afferrare in uno scenario complesso. A queste variabili, bisogna assegnargli il valore massimo e minimo, espresso in metri e rispetto al sistema di riferimento della telecamera, della regione di spazio in cui può trovarsi l'oggetto;
- *Conversione del formato dei dati della point cloud*: è un passaggio necessario, dal momento che i dati della telecamera sono memorizzati in una matrice `cv::Mat`, mentre la nostra libreria richiede in ingresso una struttura dati della libreria *PCL* (`pcl::PointCloud<pcl::PointXYZRGBA>::Ptr`). Per effettuare questa conversione, copiamo il contenuto di ogni cella della prima variabile, che sono un insieme di quattro `double`, nel corrispondente campo della variabile di tipo `pcl::PointXYZRGBA`, quindi effettuiamo il *push back* di quest'ultima variabile nella struttra dati della libreria *PCL* di nostro interesse. Il punto in esame viene inserito solo se tutti i campi della cella di partenza non sono `NaN`, altrimenti non avrebbe significato;
- *Estrazione dei soli punti appartenenti all'oggetto*: è **facoltativa**. Quest'elaborazione serve solo in scenari complessi ed utilizza le soglie impostate in precedenza. Il vettore dei punti costruito

al punto precedente viene scandito interamente, così che nella nuova struttura dati relativa alla point cloud siano contenuti solo i punti che costituiscono l'oggetto da afferare;

- *Conversione della point cloud ed estrazione dei soli punti appartenenti all'oggetto*: è **facoltativa**. In questa funzione vengono eseguiti simultaneamente gli ultimi due punti appena descritti, è stata creata solo per motivi di ottimizzazione temporale. Infatti, l'estrazione di determinati punti richiede di scandire completamente il vettore dei punti, e visto che questo è costituito all'incirca da 100.000 elementi, la scansione risulta essere troppo onerosa in termini di tempo, quindi si è scelto di creare anche una versione che svolge allo stesso tempo entrambe le elaborazioni. L'utente finale può utilizzare la versione che preferisce e commentare quella che non intende utilizzare;

- *Chiamata del plugin* `arm_grasp_gcg`: dopo che la point cloud è stata convertita nel formato richiesto ed elaborata come necessario per l'applicazione, può essere passata come parametro della funzione che richiama il plugin della libreria GPG. Il nome della variabile è determinato dall'elaborazione che si è scelta di utilizzare;

- *Salvataggio su log del vettore delle prese*: le prese ritornate dal plugin che richiama la libreria GPG vengono salvate sul log `ObjectDetection.log` nella cartella `bin`, in questo modo sono consultabili per analizzare i dati sulle prese ritornate dal plugin;

- *Creazione di un messaggio contenente il vettore delle prese*: le prese ritornate dal plugin vengono memorizzate in un messaggio di tipo `GraspCandidates` (la cui descrizione dettagliata si trova nella sezione 3.4), successivamente il messaggio viene pubblicato. Questo passaggio è utile nel caso in cui le informazioni sulle prese servono ad altre attività o componenti. All'interno del ciclo `for`, è anche possibile specificare se accettare o meno una determinata presa, per esempio imponendo delle condizioni sull'orientamento. Per fare questo, si può ottenere l'orientamento secondo la notazione RPY grazie alla funzione `getRPY` della classe `Transform3D` e imporre i vincoli desiderati.

Ogni volta in cui si apportano delle modifiche ad un'attività, è necessario ricompilare il componente scrivendo `. compile.sh` nel terminale. Invece, per eseguire il componente, bisogna scrivere `. run.sh` nel terminale. In entrambi i casi, il comando va digitato all'interno della cartella `bin` del componente d'interesse.

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. I tre campi della struttura `GraspPose` si distinguono in base al tipo di traslazione individuata:

- `coord_bottom`, contiene le coordinate della traslazione fra il sistema di riferimento della telecamera e la posizione centrata sulla base della pinza del robot;
- `coord_top`, contiene le coordinate della traslazione fra il sistema di riferimento della telecamera e la posizione centrale fra le estremità delle dita della pinza del robot;
- `coord_surface`, contiene le coordinate della traslazione fra il sistema di riferimento della telecamera e la posizione centrata sulla superficie dell'oggetto.

L'ultima struttura dati creata, `Transform3D`, è formata da 7 campi di tipo double, di cui i primi tre permettono di rappresentare le traslazioni (x, y e z, chiamati rispettivamente `a`, `b` e `c` per avere dei nomi univoci), 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/Messages
. compile.sh
```

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

Risultati

La libreria è stata testata inquadrando una bottiglietta posta ad una distanza di circa 70 cm dalla fotocamera. Di seguito, si riportano la fotografia della scena inquadrata (figura 4.1) ed il grafico delle prese individuate (figura 4.2).

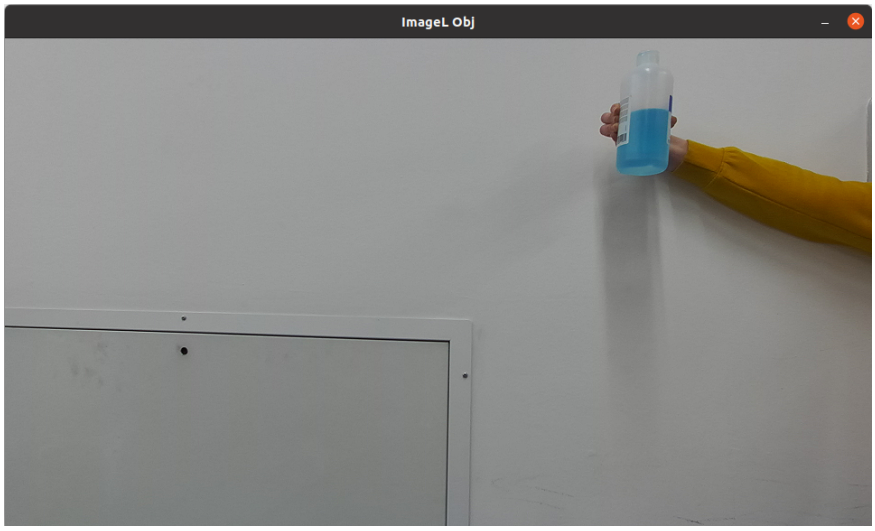

Figura 4.1: Scena inquadrata con la fotocamera stereoscopica.

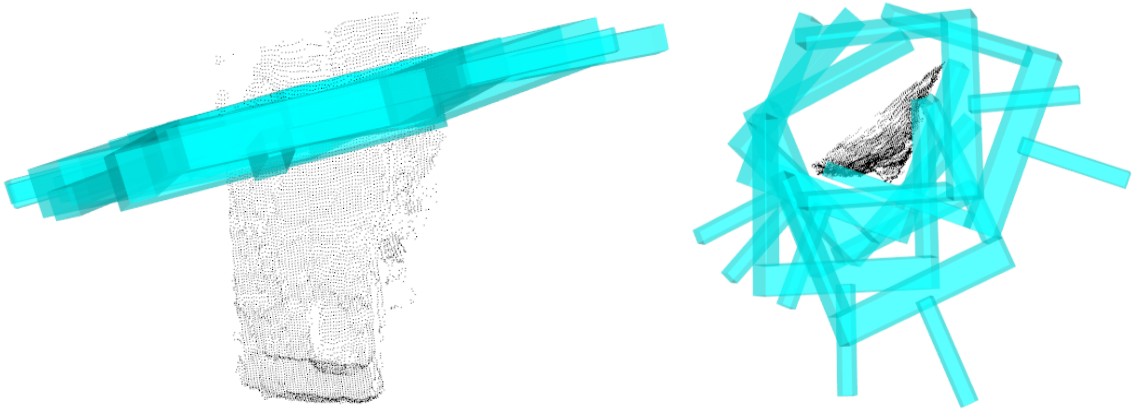

Figura 4.2: Point cloud con le prese vista frontalmente (a sinistra) e dall'alto (a destra).

Successivamente, le coordinate delle prese sono state utilizzate da un robot manipolatore a 6 gradi di libertà. In questo caso, nel file di configurazione del nostro componente è stato necessario cambiare il numero del dominio sul quale pubblichiamo il messaggio, per fare in modo che questo possa essere letto dal componente che gestisce il manipolatore. Inoltre, per semplicità di utilizzo, il nostro progetto è stato integrato in quello del manipolatore, così che si potessere gestire da un unico computer i due componenti. Di seguito, si riportano alcune fotografie dello scenario in cui è stata testato il nostro progetto.

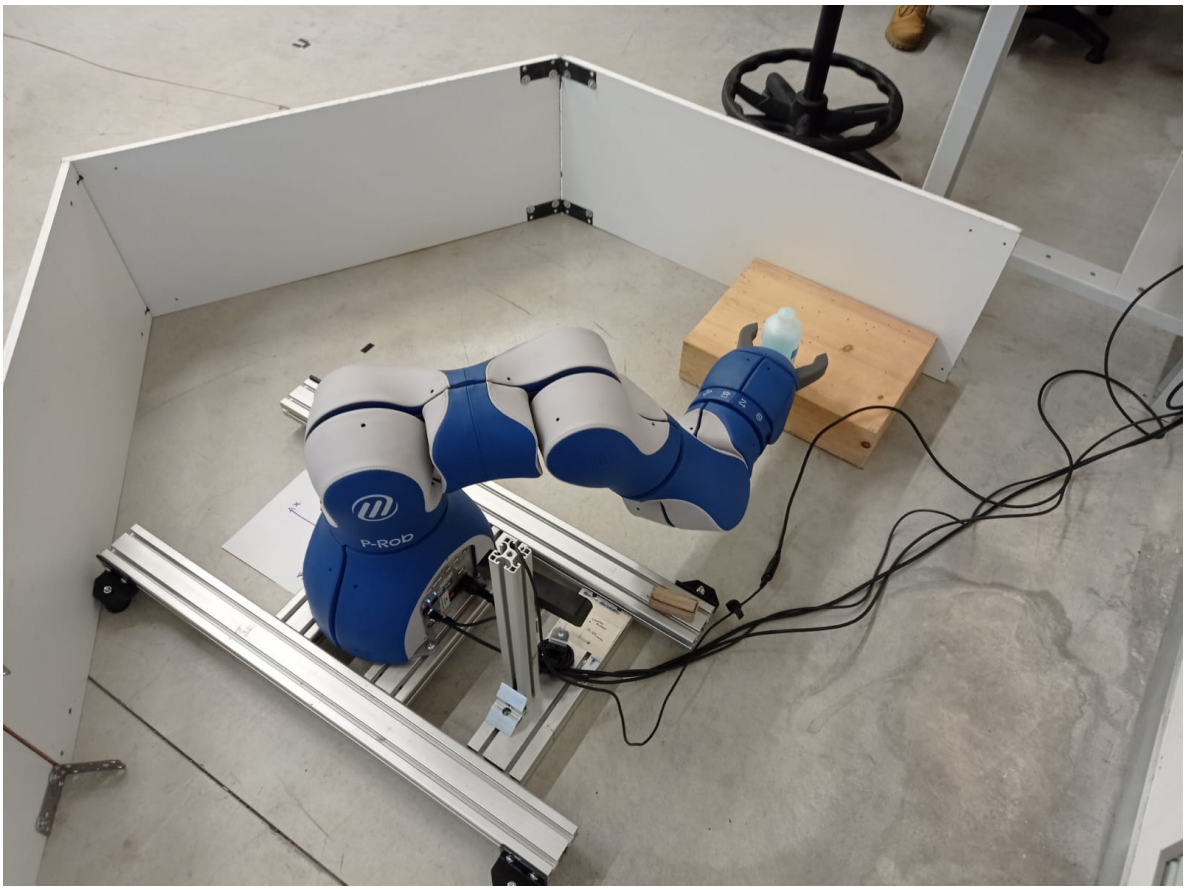

Figura 4.3: Fotografia della scena vista dall'alto.

Figura 4.4: Fotografia della scena vista lateralmente.