Report Homework Gruppo 1

Davide Pistilli, Flavio Spanò, Matteo Perin

1 Perception

1.1 AprilTag

All'avvio il programma controlla che i parametri passati come input corrispondano ai `Frame_ID` effettivi degli oggetti che è necessario rilevare e, successivamente, li converte nei loro ID numerici. Superata la fase di inizializzazione, ogni ID rilevato da *AprilTag* viene confrontato con la lista di ID richiesti dall'utente e, se viene trovata una corrispondenza, la sua pose viene codificata in un messaggio apposito e inviata sul topic `hw1_target_objects` in modo che possa essere riutilizzata da altri nodi. Gli oggetti rilevati ma non richiesti vengono ignorati, così come quelli richiesti ma non identificati da *AprilTag*.

Complessivamente, *AprilTag* si è rivelato molto semplice da utilizzare e veloce nell'esecuzione. I risultati, inoltre, hanno una buona precisione. Per questi motivi abbiamo deciso di utilizzare questo metodo per il rilevamento degli oggetti anche nelle esperienze successive.

1.2 PCL

L'utilizzo di PCL è stato, al contrario di *AprilTag*, abbastanza complesso. Per prima cosa è stato necessario filtrare la *PointCloud* in input in modo da includere unicamente gli oggetti presenti sul tavolo. Questo è stato fatto escludendo tutti i punti esterni a determinati intervalli di coordinate. A causa della leggera inclinazione del tavolo, però, i punti rilevati variano in modo significativo al variare della posizione degli oggetti. Nel caso di cubi ed esagoni la differenza non influisce sul rilevamento, cosa che invece avviene nel caso dei prismi. Questo ha portato, nei passaggi successivi, alla necessità di accorgimenti aggiuntivi.

Prima di identificare gli oggetti, però, è necessario estrarre i relativi cluster, cosa che viene fatta sulla *PointCloud* filtrata e convertita nello spazio colore HSV. I cluster vengono poi memorizzati individualmente in due versioni: con e senza l'informazione del colore. Quelli contenenti le informazioni HSV vengono utilizzati per calcolare l'altezza massima dell'oggetto e il valore medio del canale H, parametri necessari a identificare la classe a cui appartiene ogni cluster. In questa fase è stato possibile impostare le *threshold* delle altezze in modo tale da identificare correttamente gli oggetti in qualunque punto del tavolo. Per portare a termine correttamente le fasi successive, però, è stato necessario rimuovere i punti appartenenti alla base dei prismi. Questo si è reso necessario perché, al contrario degli altri oggetti, la forma dei prismi risente notevolmente della posizione all'interno del tavolo. Si è rivelato necessario,

quindi, normalizzarla in modo da ottenere risultati affidabili in ogni parte dell'area di lavoro.

Dopo aver identificato la classe dell'oggetto e aver verificato che questo è stato richiesto dall'utente (la lista viene passata come input al programma) è necessario calcolare la sua *pose*. Per fare questo abbiamo utilizzato l'algoritmo ICP, ma non direttamente sulle *PointCloud* ricevute dalla *Kinect*. A causa della loro variabilità, infatti, i risultati così ottenuti non sono sufficientemente affidabili, indipendentemente dai parametri utilizzati per il *matching*. Per ottenere delle pose adeguate, quindi, vengono appiattite le *PointCloud* dei cluster in modo che tutti i punti abbiano la stessa coordinata z e si effettua un *matching* con delle *PointCloud* bidimensionali di riferimento, create appositamente. Queste, infatti, tengono conto delle diverse densità di punti dovute all'appiattimento ed enfatizzano le caratteristiche principali degli oggetti. Con questa procedura le pose ottenute sono generalmente buone. La loro affidabilità, però, è comunque minore che con *AprilTag*, senza contare le maggiori risorse di calcolo necessarie per questo approccio. Per questo motivo abbiamo deciso di non utilizzare questo metodo nelle esperienze successive.

2 Manipulation

2.1 Posizioni dei tag

Gli oggetti *target* che il manipolatore dovrà raccogliere sono specificati in un nodo presente nell'*homework 1* che pubblica costantemente le *Pose* su un *topic*, il quale è aggiornato costantemente nel caso di eventuali rilevazioni errate o se l'oggetto viene occluso dal manipolatore (in questo caso viene inserito successivamente). Le pose rilevate dalla *Kinect* non possono essere utilizzate dal manipolatore perché sono relative a un sistema di riferimento diverso. Utilizzando la funzione *lookupTransform*, viene cambiato il sistema di riferimento delle *Pose* in modo che possano essere utilizzate correttamente dal manipolatore, che si appresterà a muoversi per raccogliere gli oggetti sul tavolo da lavoro.

Al fine di evitare collisioni tra manipolatore ed oggetti è stata inizializzata la *Collision Detection* inserendo le *mesh* utilizzate nell'*homework 1* posizionate nello spazio di collisione tramite i rilevamenti della *Kinect*.

2.2 Posizione del manipolatore

Nella programmazione del manipolatore è stato scelto di spostare il robot in due modi: tramite spostamenti con i giunti oppure in cartesiano. Questa combinazione di movimenti ha permesso di rendere il robot più stabile ed evitare improvvisi movimenti che potrebbero portare a collisioni inaspettate.

Questo tipo di manipolatore presenta 5 giunti, che possono essere inizializzati a dei valori ben precisi ma, durante le prove fatte in laboratorio, più i valori risultavano essere precisi (aumentandone i decimali), più il manipolatore si arrestava bruscamente.

Letto il contenuto del *topic* contenente le *pose* dei vari oggetti posti nell'area di lavoro, il manipolatore si reca nella posizione di riferimento al di sopra dell'area di lavoro, tramite uno spostamento di giunti. Successivamente, il manipolatore si sposterà sopra l'oggetto che si è scelto di prendere con un'altezza pari ad: altezza del tavolo + altezza dell'oggetto + offset. Una volta ultimato il posizionamento sopra l'oggetto, si scende di metà offset. Dopo essere giunti in questa posizione si è notato, durante le fasi di test sull'avvicinamento degli oggetti, che il piano di lavoro risulta essere leggermente inclinato. Ciò rende la procedura più complessa poiché, per ogni oggetto posto in aree diverse del piano di lavoro, la discesa dell'end-effector deve essere maggiore nella parte di destra e minore nella parte sinistra. A causa di questo problema è stata implementata un'equazione che permette al manipolatore di scendere fino al valore corretto, il quale rappresenta la posizione dell'apriltag in base alla posizione dell'oggetto:

$$\text{TABLE\_INCLINATION} \cdot (-\text{currentPose.pose.position.x} + \text{TABLE\_MAX\_X})$$

Infine, viene utilizzato l'approccio cartesiano per effettuare il grasping sull'oggetto, il quale calcola un percorso cartesiano in una sola dimensione che determina uno spostamento (in metri) tra le configurazioni dell'*end-effector* e la posizione dei risultati finali. La funzione restituisce un valore compreso tra 0.0 e 1.0, che indica la percentuale di successo del percorso calcolato, oppure -1.0 in caso di errore. È stato imposto che se il valore restituito è maggiore di 0.8, allora si procede con l'avvicinamento all'oggetto. Questo approccio è però valido solo per il cubo e l'esagono, perché l'*apriltag* è posizionato al di sopra dell'oggetto senza presentare alcuna inclinazione. Nel caso del prisma, l'*apriltag* è situato in uno dei lati inclinati, in questo caso si deve ruotare l'end-effector in modo da farlo posizionare parallelamente al *tag*. Viene quindi eseguito un cambio di coordinate dalle *pose* (relative alla camera) in quelle rispetto all'*end-effector*, in questo modo il manipolatore si posizionerà in modo parallelo all'apriltag. Successivamente si avvicinerà come descritto precedentemente per agganciare l'oggetto e spostarlo nella posizione stabilita.

Una volta che il manipolatore ha effettuato l'aggancio dell'oggetto, esso ritornerà nella *reference pose* e attenderà il comando dell'utente per determinare dove posizionare l'oggetto inserendo:

1. Se si vuol sganciare l'oggetto nella *docking station 1*
2. Se si vuol sganciare l'oggetto nella *docking station 2*

Una volta inserito da terminale il numero, il manipolatore si posizionerà al di sopra della postazione designata e infine si abbasserà per rilasciare l'oggetto.

Nel caso dell'avvio del programma in simulazione, troviamo alcune differenze rispetto al reale. Una di queste è l'aggancio dell'oggetto, che in simulazione prevede un gripper anziché una calamita. Questo rende più facile l'aggancio dell'oggetto, non dovendo considerare la rotazione dell'*end-effector* nel caso del prisma.

3 Navigation

La sfida principale di questa esperienza è stata la configurazione dei parametri di navigazione per permettere il corretto svolgimento del percorso all'interno dell'arena. Come obiettivo addizionale si è cercato di portare a termine il corridoio stretto iniziale facendo uso soltanto del *navigation stack* di ROS, mentre le scansioni laser sono state utilizzate per controllare quali sono i passaggi aperti e per alcuni controlli aggiuntivi delle collisioni.

Utilizzando la configurazione di default si è potuto notare che il corridoio iniziale non poteva essere percorso, dato che Marrtino si bloccava dopo pochi centimetri. Il problema principale stava nella pianificazione globale che non manteneva un percorso parallelo al corridoio: durante il *local planning* ci si poteva trovare quindi in una posizione troppo vicina a una delle due pareti, attivando il meccanismo di collisione e impedendo perciò il proseguimento della navigazione. Per ovviare a questa situazione abbiamo aumentato l'inflazione della mappa globale in modo tale da rendere la pianificazione il più possibile parallela alle pareti e diminuito l'inflazione della *costmap* locale per evitare che il *planner* rilevasse una collisione troppo in anticipo. I valori impostati hanno permesso il completamento del passaggio in modo corretto.

La successiva complicazione è stata la curva stretta per entrare nell'arena. La pianificazione cercava sempre di eseguire una curva troppo larga, abbiamo quindi cercato di risolvere il problema cambiando alcuni parametri:

- Leggera diminuzione ($\Delta = 0.025$) del footprint orizzontale di Marrtino per minimizzare le false collisioni. Questo cambiamento non ha portato all'incremento delle collisioni, ma ha permesso un maggiore spazio di manovra nel corridoio iniziale e durante la curva.
- Aumento della risoluzione delle misure laser. Ciò ha permesso un migliore rilevamento degli ostacoli e della localizzazione, sempre nell'ottica di diminuire la possibilità di uno stato di collisione prematuro.
- Aumento di *path distance bias* per evitare che il *local planner* andasse troppo fuori dal percorso globale e verso una situazione che provoca il fallimento della pianificazione, data la ristrettezza dello spazio di movimento.
- Diminuzione di *goal distance bias* per evitare che la pianificazione fallisse nonostante fosse stata raggiunta la posizione desiderata con un errore accettabile.
- Aumento delle velocità massime di rotazione e traslazione per permettere una traiettoria curvilinea più stretta.
- Impostazione velocità minima a una valore negativo, per permettere anche la possibilità di retromarcia nel caso di necessità.

Nonostante questi accorgimenti non è comunque stato possibile ottenere una pianificazione che riuscisse a completare la curva in un modo adeguato, in quanto si arrivava sempre a una situazione in cui il robot si trovava troppo vicino al muro.

Per risolvere il problema è stata introdotta una procedura di retromarcia nel caso fosse impossibile procedere con la pianificazione. La distanza percorsa è incrementata ulteriormente se non si riesce a proseguire. Per evitare collisioni vengono utilizzate le scansioni laser e il tentativo di *backing-off* viene soppresso nel caso ci si trovi troppo vicino a una parete. Questa procedura ha permesso di completare la curva ed entrare nell'arena.

Il rilevamento di ostacoli viene effettuato tramite una funzione che controlla se in un angolo di fronte allo scanner (ricevuto in input) ci sono almeno dieci scansioni con un *range* entro una certa soglia. Se ci sono abbastanza punti vicini allo scanner viene quindi rilevata la presenza di un ostacolo.

Grazie a tutti gli accorgimenti citati è stato possibile portare a termine il percorso, che segue questi passi:

- Inizio del movimento con punto di arrivo impostato presso i *gate* di ingresso nell'arena. La direzione di arrivo è parallela a *corridor 1*.
- Verifica del *gate* aperto tramite la funzione di rilevamento: se non viene rilevato un ostacolo davanti alla posizione attuale significa che è aperto *gate 2*, altrimenti l'entrata libera è *gate 1*.
- Se l'entrata libera è *gate 2* si procede fino alla fine di *corridor 1*.
- Movimento fino alla *docking station 1* per verifica dell'eventuale ostacolo.
- Spostamento verso *docking station 2* se viene rilevato l'ostacolo, altrimenti avanzamento verso *docking station 1*.
- Movimento verso l'entrata di *corridor 2*.
- Movimento verso la fine di *corridor 2*.
- Movimento verso *start* (fine del percorso).