Università degli Studi di Padova

Tesi Magistrale in Ingegneria Informatica

Object Detection e Visual Servoing per applicazioni robotiche di grasping e manipolazione

---

Relatore:
Prof. Stefano Ghidoni
Correlatore:
Ph.D. Roberto Bortoletto

Studente Magistrale:
Silvia Gandin

Intelligent Autonomous Systems Laboratory (IAS-Lab)
Department of Information Engineering (DEI)

11 Aprile 2017

Ringraziamenti

Vorrei ringraziare lo IAS-Lab, con cui ho lavorato in questi mesi, e tutta la squadra Desert Lion con cui ho condiviso questa avventura. Un grazie speciale alla mia famiglia e ai miei amici per essermi stati vicino e avermi sostenuto in questo percorso universitario.

Sommario

Object Detection e Visual Servoing per applicazioni robotiche di grasping e manipolazione

Il lavoro di questa tesi ha preso spunto dalla Challenge MBZIRC per coprire i seguenti ambiti: *Object Detection* di utensili riflettenti, *3D Pose Estimation*, *Object Tracking* e *Visual Servoing*. Per localizzare gli oggetti sono stati creati dei *boosted cascade classifier* che anche in condizioni di luce intensa, riflessi e ombre, hanno restituito ottimi risultati (precisione maggiore del 90%). Sono stati sviluppati inoltre programmi in grado di determinare la posizione e l'orientazione 3D degli oggetti, tramite la Trasformata di Hough e il pacchetto `tf`. E' stato infine implementato un programma in grado di tracciare un oggetto in una sequenza di immagini, fornendo il feedback visivo per muovere il robot di conseguenza, grazie al visual servoing.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La crescita esponenziale nel campo della robotica è sotto gli occhi di tutti, e in parallelo l'intelligenza artificiale sta ottenendo risultati sempre più rilevanti. Il test di Turing è diventato obsoleto, mentre software vocali di intelligenza artificiale sono ormai presenti in tutti gli smartphone. I video con cui la Boston Dynamics, divisione di Google dedicata allo sviluppo di robot avanzati, mostra le sue nuove creazioni sono pubblicati e condivisi da migliaia di persone stupefatte ed entusiaste. Può sembrare un paradosso, ma è proprio la natura l'osservata speciale della robotica. Quale miglior esempio infatti, se non il frutto di miliardi di anni di evoluzione? E così, per creare il robot più veloce al mondo, la Boston Dynamics non ha potuto che ispirarsi ad un ghepardo, creando Cheetah[8] (Figura 1.1).

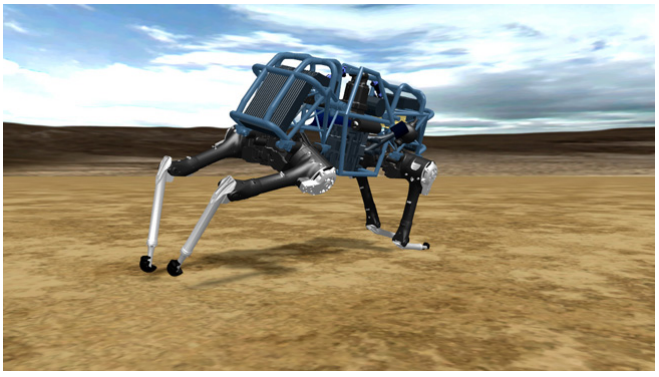


FIGURA 1.1: Il robot Cheetah della Boston Dynamics, ispirato ad un ghepardo.

Ma la robotica non consiste solo nel creare robot veloci e robusti, in grado di compiere azioni magari ripetitive: la vera sfida è unire alla robotica l'intelligenza artificiale. E come può un robot avvicinarsi ad un principio di intelligenza, se non è in grado di

percepire il mondo esterno? Da qui, l'importanza sempre maggiore che sta acquisendo la *computer vision*.

In questa tesi si è in particolare trattato della tematica dell'**object detection**, forse la sfida più difficile della computer vision in questi e nei prossimi anni. Non si tratta solo di localizzare un oggetto nell'immagine o video, ma anche e soprattutto di riconoscerlo, assegnargli un nome. Come si insegna ad un robot cos'è un uomo, un cane, un albero? Come può riconoscerli in un'immagine, che per un programma è solo un insieme di numeri? Non è ancora stata trovata una risposta completa, e nonostante i notevoli progressi l'object detection è ancora una sfida incompiuta per la computer vision.

Un altro argomento trattato è la **visione 3D**, che solo negli ultimi anni ha potuto essere studiata su larga scala a causa dei notevoli costi computazionali. In particolare si è studiato come stimare la posizione nello spazio di un oggetto, requisito fondamentale per controllare un robot e farlo interagire con altri oggetti.

L'ultimo ambito toccato in questa tesi riguarda l'**object tracking** e il **visual servoing**. Queste due tecniche lavorano in sinergia per permettere il movimento di un sistema robotico in base ad un feedback visivo. Più precisamente, rendono possibile il tracking di un oggetto in una sequenza di immagini o video, e calcolano come muovere il robot per raggiungerlo o seguirlo.

1.1 MBZIRC - Challenge 2

L'Università di Padova, ed in particolare lo IAS-Lab[1], hanno preso parte alla competizione internazione di robotica organizzata ad Abu Dhabi (Emirati Arabi Uniti) nel Marzo 2017 dalla Khalifa University[2].

MBZIRC (Mohamed Bin Zayed Internation Robitic Competition) è stata suddivisa in tre challenge: la prima e la terza richiedevano l'utilizzo di droni, la seconda di un robot mobile manipolatore. La nostra squadra Desert Lion, che ha partecipato a quest'ultima challenge, si è iscritta alla competizione insieme ad altre 145 squadre, ed è riuscita a rientrare nelle 28 finaliste per la gara finale ad Abu Dhabi.

[1] Intelligent Autonomous Systems Laboratory: http://robotics.dei.unipd.it/
[2] Khalifa University: http://www.kustar.ac.ae/

FIGURA 1.2: Mohamed Bin Zayed Internation Robotic Competition.

La challenge 2 richiedeva l'utilizzo di un UGV (*Unmanned Ground Vehicle*) per localizzare e raggiungere un pannello in un'arena. Su questo pannello erano appese sei chiavi combinate, ed era fissata una valvola. Il robot doveva prendere la chiave delle dimensioni giuste per essere inserita nella valvola, e ruotare quest'ultima di 360 gradi.

La tesi svolta ha sfruttato l'opportunità fornita da MBZIRC per sviluppare dei task di visione, cercando soluzioni specifiche per la challenge che potessero tuttavia essere utili per altre applicazioni. Gli obiettivi di questa tesi hanno riguardato:

1. *La localizzazione delle chiavi.*
2. *L'individuazione della chiave corretta.*
3. *La stima della sua posizione 3D e orientazione per il grasping.*
4. *La localizzazione della valvola.*
5. *La stima della sua posizione 3D e orientazione per l'inserimento.*

1.2 Panoramica della tesi

Questa tesi si suddivide in sei capitoli:

Capitolo 1 - *Introduzione* contestualizza il ruolo della robotica e della computer vision al giorno d'oggi, e descrive la MBZIRC Challenge con particolare riguardo alle tematiche di questa tesi.

Capitolo 2 - *Setup Sperimentale* elenca l'hardware e il software utilizzato nel corso del lavoro di tesi.

Capitolo 3 - *Object Detection di utensili riflettenti* illustra lo stato dell'arte ed l'approccio scelto per identificare gli oggetti della Challenge, mostrando i risultati ottenuti.

Capitolo 4 - *Stima della posa 3D per presa ed inserimento* descrive la visione 3D e i metodi utilizzati per localizzare nello spazio gli oggetti.

Capitolo 5 - *Object Tracking e Visual Servoing* illustra lo studio ed i test effettuati per una possibile futura implementazione del movimento del braccio manipolatore in base a feedback visivo.

Capitolo 6 - *Conclusioni* sintetizza il contributo della tesi e e le sue applicazioni generali.

Capitolo 2

Setup Sperimentale

2.1 RUR53

2.1.1 Overview

Il robot assemblato per la challenge (Figura 2.1) è costituito da numerose parti.

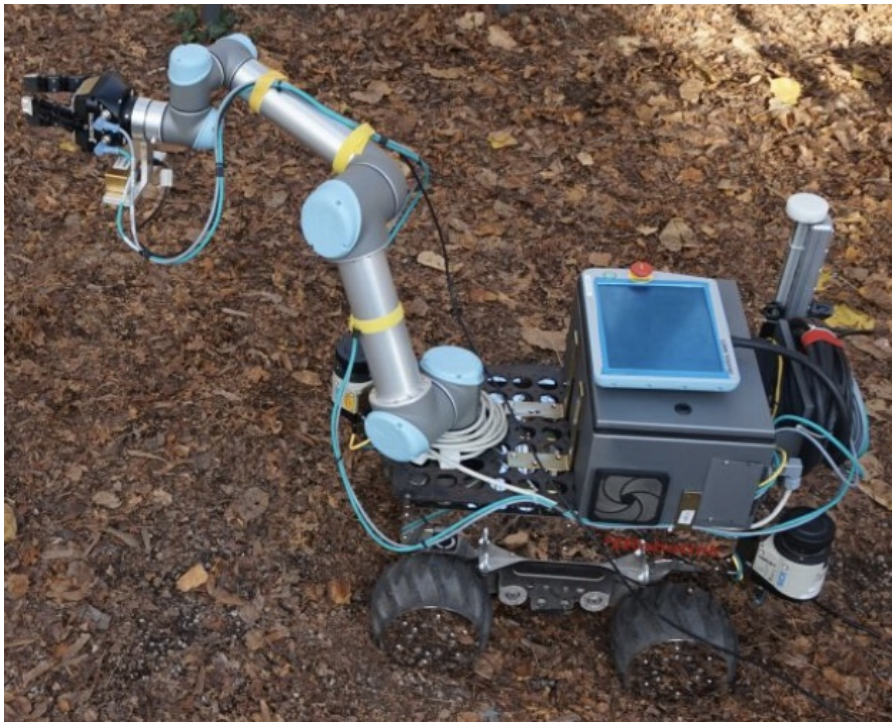


Figura 2.1: Il robot RUR53, utilizzato per MBZIRC.

Innanzitutto è presente una base mobile outdoor Summit XL HL (Robotnik Automation), con quattro ruote motrici, su cui è montato un braccio manipolatore UR5 (Universal Robots). All'estremità del braccio è stato aggiunto un gripper, il Robotiq Adaptive Gripper a tre dita (Robotiq), e il sistema di visione stereo, inizialmente composto da una stereo camera BumbleeBee 2 (Point Grey Research), sostituita da due Grasshopper 3 (Point Grey Research). Sono presenti infine due 2D laser scanner (SICK Sensor Intelligence).

Il nome assegnato, RUR53, deriva dai nomi dei produttori (Robotnik, Universal robots, Robotiq), con il numero 5 da UR5 e il 3 dal gripper a tre dita.

2.1.2 Sensori ottici

Come sensori visivi sono stati utilizzati due sistemi:

1. **BumbleBee2 versione 1394a**[1] (Figura 2.2): questa stereo camera contiene sensori ottici a colori con risoluzione 1032x776 pixel e frame rate di 20 FPS, e permette una visione 3D della scena.

FIGURA 2.2: La stereo camera BumbleBee 2.

2. **Due Grasshopper3 2.8MP Mono USB3**[2] (Figura 2.3): tramite queste due telecamere, con risoluzione di 1920x1440 pixel e frame di 26 FPS, si è ottenuto un sistema di visione stereo 4, in grado di elaborare informazioni tridimensionali della scena.

[1] Scheda tecnica BumbleBee: https://www.ptgrey.com/bumblebee2-stereo-vision-08-mp-color-firewire-1394a-25mm-sony-icx204-camera

[2] Scheda tecnica Grasshopper: https://www.ptgrey.com/grasshopper3-28-mp-mono-usb3-vision-sony-icx674-camera

FIGURA 2.3: La telecamera Grasshopper 3.

2.1.3 UR5

Il braccio manipolatore scelto è l'**UR5** della Universal Robots[3] (Figura 2.4). Leggero e flessibile, ha un raggio d'azione fino a 850 mm grazie ai suoi sei giunti e un payload di 5 kg. Tramite il monitor touchscreen è possibile operare con il robot con un'interfaccia semplice ed intuitiva.

FIGURA 2.4: Il braccio manipolatore UR5.

[3]Scheda tecnica UR5: https://www.universal-robots.com/it/prodotti/robot-ur5/

2.2 Software

2.2.1 ROS

ROS (Robot Operating System)[4] (Figura 2.5) è un insieme di librerie e strumenti che permettono la creazione di applicazioni robotiche. Viene definito sistema operativo in quanto fornisce funzionalità avanzate come astrazione dell'hardware, driver, librerie, strumenti di visualizzazione, comunicazione tra processi. ROS è rilasciato sotto licenza open source, e supporta due linguaggi di programmazione: C++ e Phyton.

FIGURA 2.5: Logo di ROS.

ROS fa della modularità uno dei suoi punti di forza: i processi sono suddivisi in nodi, che comunicano tra loro tramite servizi, azioni o topic. Questi ultimi in particolare costituiscono un canale di comunicazione particolarmente utilizzato quando si lavora con i robot, permettendo di diffondere messaggi come immagini o comandi.

2.2.2 OpenCV

OpenCV (open source Computer Vision)[5] (Figura 2.6) è una libreria software di visione artificiale e machine learning. Fornisce l'implementazione di più di 2500 algoritmi, che possono essere utilizzati per localizzare e riconoscere oggetti, seguirne i movimenti, elaborare le immagini, produrre dati 3D ecc.

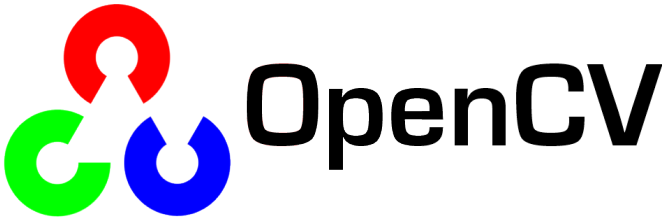


FIGURA 2.6: Logo di OpenCV.

[4]ROS: http://www.ros.org/
[5]OpenCV: http://opencv.org/

OpenCV è rilasciato sotto licenza BSD ed è gratuito per un utilizzo sia accademico sia commerciale. Supporta i linguaggi di programmazione C++, C, Phyton e Java ed ha un'interfaccia per Windows, Linux, Max OS, iOS e Android. OpenCV è stato sviluppato per garantire efficienza computazionale, con particolare attenzione ad applicazioni real-time.

Adottato in tutto il mondo, OpenCV ha più di 50 mila utenti ed è stato scaricato più di 14 milioni di volte.

2.2.3 ViSP

ViSP (Visual Servoing Platform)[6] (Figura 2.7) è una libreria open-source scritta in C++ che fornisce un utile insieme di strumenti per applicazioni di object tracking e visual servoing. E' una libreria modulare e cross-platform, e implementa più di 270 classi documentate. Ulteriori informazioni sono fornite nel Capitolo 5.

Figura 2.7: Logo di ViSP.

2.2.4 Ambienti di simulazione

In ogni applicazione robotica, fondamentale è il testing in un ambiente di simulazione dedicato.

I due programmi utilizzati in questa tesi sono:

1. **Gazebo**[7] (Figura 2.8): simulatore gratuito che rende possibile progettare robot e testare algoritmi utilizzando uno scenario realistico. Gazebo è in grado di simulare efficacemente popolazioni di robot in un ambiente complesso, fornendo un robusto motore fisico.

[6] ViSP: https://visp.inria.fr/
[7] Gazebo: http://gazebosim.org/

FIGURA 2.8: Logo di Gazebo.

2. **V-Rep**[8] (Figura 2.9): è un simulatore robotico basato su un'architettura distribuita molto versatile. E' più robusto e potente di Gazebo, offrendo più funzionalità e API.

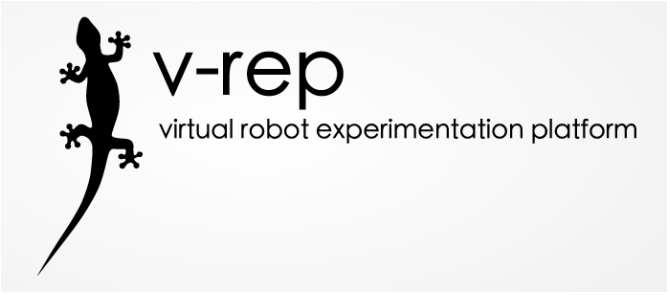


FIGURA 2.9: Logo di V-Rep.

[8]V-Rep: http://www.coppeliarobotics.com/

Capitolo 3

Object Detection di utensili riflettenti

3.1 Stato dell'arte

L'**object detection** nel campo della computer vision è il processo mediante il quale si localizzano e identificano una o più classi di oggetti in una sequenza di immagini o video. Ciò che per un essere umano appare un compito semplice ed intuitivo si rivela in realtà una delle sfide più impegnative e appassionanti per la visione artificiale. Si prenda in considerazione il seguente esempio: un uomo, anche un bambino, è in grado di riconoscere senza nessuno sforzo un cane in un'immagine. Per un computer, un'immagine non è altro che una griglia di numeri, e trovare un legame tra una particolare serie di numeri e la figura di un cane risulta essere un compito estremamente complesso. Ora si pensi a quante razze di cani, anche notevolmente differenti, siano presenti in natura, e in quante posizioni possano apparire nelle immagini, con diverse condizioni di luce e ambientali: forse così si può iniziare a comprendere appieno quanto l'object detection sia una sfida articolata ed impegnativa.

Nell'ambito dell'object detection due termini acquisiscono fondamentale importanza:

- **Machine Learning**: è il processo mediante il quale si trasformano i dati in informazione estraendo regole o pattern.

- **Classification**: è una categoria di algoritmi di machine learning che impara a riconoscere una particolare tipologia di oggetti, assegnando delle etichette (*label*).

Un generico *image classifier* richiede i seguenti passaggi (Figura 3.1)[18]:

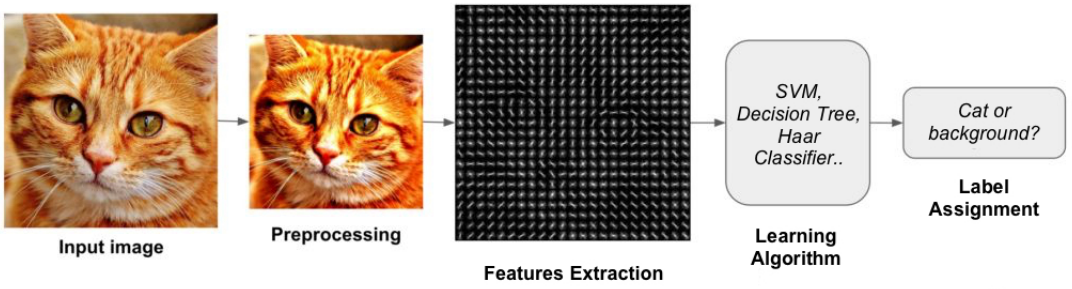


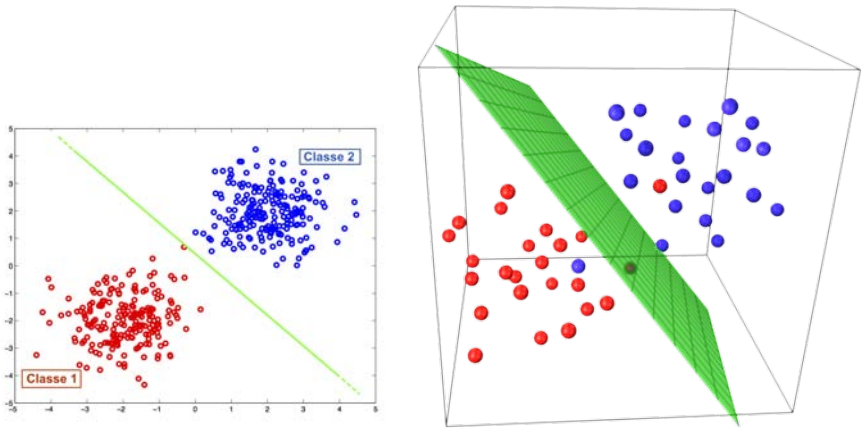
FIGURA 3.1: Image classifier.

1. **Preprocessing**: generalmente l'immagine di input viene modificata, ad esempio per ridurre il rumore o le dimensioni. Spesso viene normalizzato il contrasto e la luminosità, e applicato un filtro di sfocatura; un altro step di pre-processing piuttosto comune è la trasformazione di un'immagine a colori in una nella scala di grigi. In altri casi viene applicato un edge-detector, che restituisce un'immagine binaria con i contorni.

2. **Feature extraction**: le immagini generalmente contengono molta più informazione di quanto sia necessario per la classificazione. Per questo vengono estratte e mantenute solo le *feature*, ossia delle caratteristiche rilevanti dell'immagine, su cui verrà effettuato il processo di *learning* successivo. Alcune feature molto utilizzate sono le *Haar* (Figura 3.4), le *Local Binary Pattern* (Figura 3.5), le *Scale-Invariant Feature Transform* (SIFT)[16], le *Speeded Up Robust Feature* (SURF)[1] e gli *Histogram of Oriented Gradients* (HOG)[5]. Non esiste una sola tipologia di *feature* migliore delle altre, esse vanno scelte in funzione del task da effettuare, dato che ognuna presenta delle caratteristiche diverse.

3. **Learning Algorithm**: il processo di *feature extraction* restituisce un vettore di *feature*, che viene ricevuto in input dal programma di classificazione. Per essere in grado di identificare un oggetto, gli algoritmi di machine learning richiedono

la costruzione di un dataset, diviso in un corposo training set e in test set più ridotto. Il training set viene utilizzato dall'algoritmo per imparare delle regole su cui costruire il modello finale; il test set serve invece per verificare le performance del programma di previsione. Ci sono molteplici algoritmi di learning, ad esempio le famose *Support Vector Machines* (SVM)[4] e i più semplici *Decision Trees*. Il principio generale alla base della maggior parte di essi è quello di considerare i vettori di *feature* come punti nello spazio a più dimensioni, e di individuare piani o superfici che suddividano lo spazio, separando i punti di classi diversi (Figura 3.2).



(a) (b)

FIGURA 3.2: SVM: suddivisione dello spazio con una retta nel 2D (a) e con un piano nel 3D (b).

4. **Label Assignment**: per assegnare un'etichetta ad un'immagine si prende il suo vettore di *feature*, trasformato in un punto nello spazio. Utilizzando il delimitatore (retta, piano o iperpiano) restituito dal processo di training, si verifica in quale posizione si trovi il punto e quindi a che classe appartenga l'oggetto.

Boosted Cascade Classifier

I *boosted cascade classifier* sono una particolare tipologia di object detector basati sulla tecnica *cascading*, che sfrutta una successione di stage per raggiungere progressivamente gli obiettivi di detection desiderati[7]. Un boosted cascade classifier è quindi costruito gradualmente con la concatenazione di numerosi classificatori più semplici (Figura 3.3): per ogni stage i risultati dei classificatori deboli precedenti vengono utilizzati come punto

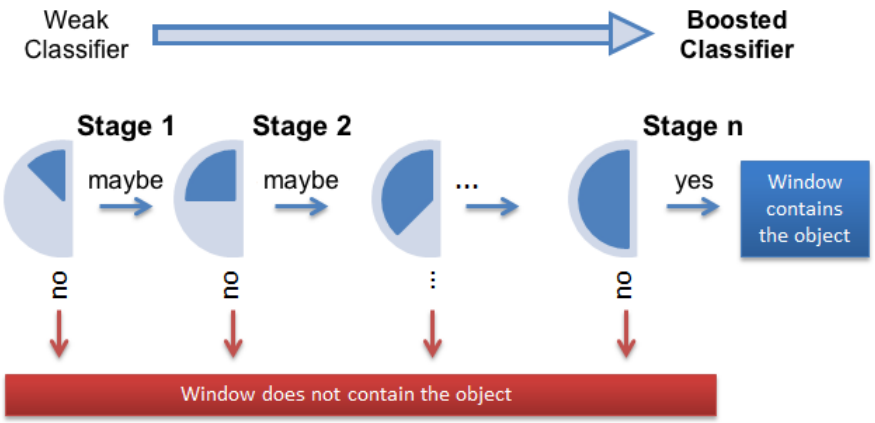


FIGURA 3.3: Cascade Classifier.

di partenza per l'analisi successiva. La tecnica di learning alla base di questo processo è chiamata *boosting*, e permette appunto di allenare iterativamente classificatori deboli per poi unirli, opportunamente pesati in base alla loro precisione, in un classificatore finale. Infatti, per quanto poco precisi siano i classificatori deboli, finché risultano anche solo di poco migliori della scelta casuale ($error\ rate < 0.5$ per un problema di classificazione binaria) il modello converge in un classificatore finale più forte.
Un boosted cascade classifier si può esprimere nella seguente forma matematica:

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \ ,$$

dove ogni f_t è un classificatore debole che riceve in input un'immagine x e restituisce in output, in caso di un problema binario, un valore di segno positivo o negativo a seconda della classe, e indicante la confidenza della classificazione. Il classificatore finale F_T avrà quindi valore positivo se l'oggetto viene assegnato alla classe positiva, negativo in caso contrario.

I primi algoritmi boosting furono proposti a partire dal 1990, ma il più famoso e tuttora diffusamente utilizzato, **AdaBoost**, venne sviluppato nel 1997 da *Robert Schapire* e *Yoav Freund* [10]. AdaBoost (*Adaptive Boosting*) fu il primo algoritmo di boosting ad essere adattivo, riuscendo a focalizzarsi sugli errori di riconoscimento dei precedenti classificatori per cercare di ottenere un migliore risultato negli stage seguenti. AdaBoost venne utilizzato anche nel modello proposto da *Paul Viola* e *Michael Jones* [26]. Viola e Jones furono tra i primi a sviluppare un object detector sfruttando le tecniche di cascading e di boosting, e tuttora il loro metodo viene preso ad esempio ed utilizzato

con vari miglioramenti. Il loro classificatore si basava inoltre sulle **Haar-feature**, che vengono estratte dalle immagini e analizzate nei vari stage. Una Haar-feature (Figura 3.4) è un *visual descriptor*, rappresentato da un valore numerico ottenuto sottraendo la somma dei pixel sottostanti l'area bianca dalla somma dei pixel sotto quella nera.

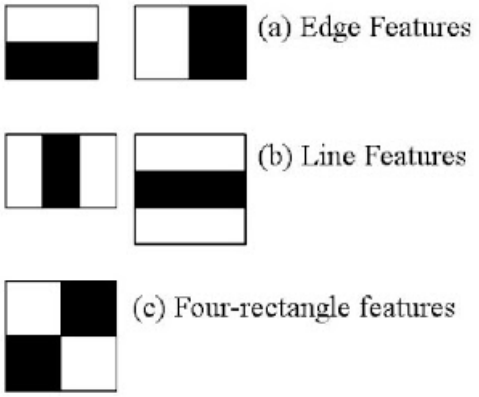


FIGURA 3.4: Haar-features.

Un altro esempio di visual feature che attualmente vengono spesso preferite alle Haar sono le **Local Binary Patterns**. Le *LBP* feature, descritte per la prima volta nel 1994 da *T. Ojala, M. Pietikäinen*, e *D. Harwood* [21], sono dei semplici numeri interi che rappresentano l'intensità di un intorno di pixel. Per ottenere un vettore di LBP feature si suddivide l'immagine in blocchi, a loro volta costituiti da celle di dimensione 3×3 pixel. Considerando come soglia (*threshold*) il valore del pixel centrale, si effettua un confronto in senso orario o antiorario con i pixel adiacenti, con risultato 0 se il valore centrale è maggiore, 1 viceversa. Si ottiene così un numero binario di 8 cifre, da convertire in decimale (Figura 3.5). Questo processo viene ripetuto per ogni pixel del blocco, in modo da costruire un istogramma con le frequenze di occorrenza di ciascun numero decimale risultante. Gli istogrammi ottenuti da ogni blocco vengono infine concatenati in un istogramma finale per l'intera immagine(Figura 3.6).

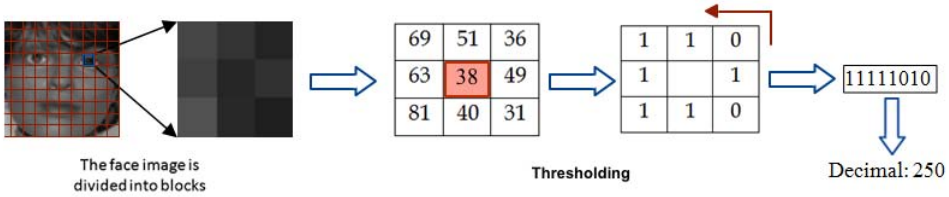


FIGURA 3.5: LBP Thresholding.

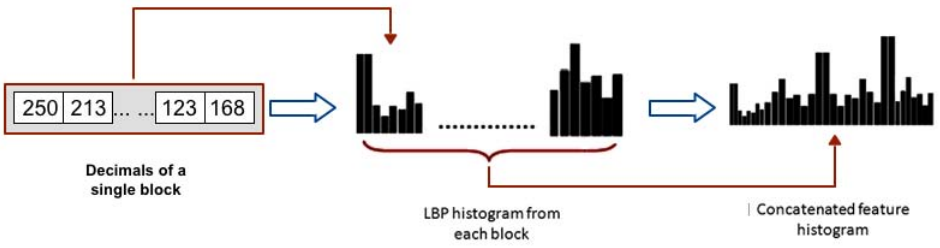


FIGURA 3.6: LBP Histograms.

Una volta scelte le visual feature che si vogliono utilizzare, si può procedere con la creazione del classificatore. Il processo di sviluppo e utilizzo di un boosted cascade classifier si può suddividere in tre fasi principali:

1. **Creazione di un dataset**: inizialmente è necessario acquisire un numero di immagini consistente con gli obiettivi che si vogliono raggiungere. Se il task è limitato ad una particolare situazione o l'oggetto da identificare ha caratteristiche semplici e ben precise (ad esempio un logo bidimensionale), possono risultare sufficienti poche centinaia di immagini, viceversa se si vuole ottenere un classificatore più robusto e utilizzabile in numerose circostanze il dataset può essere costituito anche da migliaia di acquisizioni. Il dataset è suddiviso in immagini che rappresentano l'oggetto da identificare, chiamate **positive samples**, e immagini che non rappresentano l'oggetto, chiamate **negative** o **background samples**.

2. **Training**: creato il dataset desiderato si può procedere alla fase in cui viene concretamente costruito il classificatore. Dalle immagini del dataset vengono estratte delle feature: la tecnica di boosting associata al cascading si rivela fondamentale in questa fase in quanto il numero di feature viene ridotto considerevolmente. Vengono infatti mantenute solo quelle utili e distribuite nei vari stage, permettendo un notevole miglioramento in termini di efficienza e velocità.

3. **Detection**: terminato il training, il classificatore è pronto per essere utilizzato su nuove immagini. Il processo di *detection* consiste nel creare una *sliding window* che percorre l'intera immagine e su cui viene applicato il classificatore. Quest'ultimo restituirà il valore 1 se in quella parte di immagine è probabile ci sia l'oggetto da

identificare, altrimenti restituirà il valore 0. Il classificatore è facilmente ridimensionabile, in modo da poter cercare l'oggetto nell'intera immagine senza che le sue dimensioni siano note in precedenza.

3.2 Approccio utilizzato per MBZIRC

La challenge a cui si è partecipato richiedeva come compito primario l'identificazione di un set di **chiavi combinate**, sospese a dei pioli, e di una **valvola**, di forma nota. Il problema principale, che ha limitato notevolmente la scelta di object detector utilizzabili, riguardava le caratteristiche dei materiali degli oggetti da identificare. Infatti sia le chiavi sia la valvola, essendo metallici, presentano numerosi riflessi di luce, resi ancora più rilevanti dall'ambientazione outdoor della competizione. L'impossibilità di prevedere con esattezza la situazione ambientale, l'inclinazione e l'intensità della luce ha imposto di cercare un approccio il più stabile e robusto possibile a variazioni di luce. Un altro problema riguardava la forma stessa degli oggetti: secondo specifiche le chiavi combinate seguivano lo standard **ISO 7738**, il quale impone solo alcuni vincoli lasciando variabili numerosi parametri.

Secondo la descrizione ufficiale dello standard ISO 7738[12], una chiave combinata presenta le seguenti caratteristiche (Figura 3.7):

- Forma: *offset A* o *offset B.*
- Serie: *short, medium* o *long.*
- Apertura della chiave:s, in millimetri.
- Lunghezza totale: l, in millimetri.
- Spessore della testa: e, in millimetri.
- Inclinazione della testa: $15° \pm 5°$.

Lo stelo della chiave tuttavia può cambiare forma a seconda del set scelto, con un effetto di bombatura più o meno evidente e differenti lunghezze. Come si può vedere nella figura 3.8 quest'ultimo parametro in particolare può variare notevolmente.

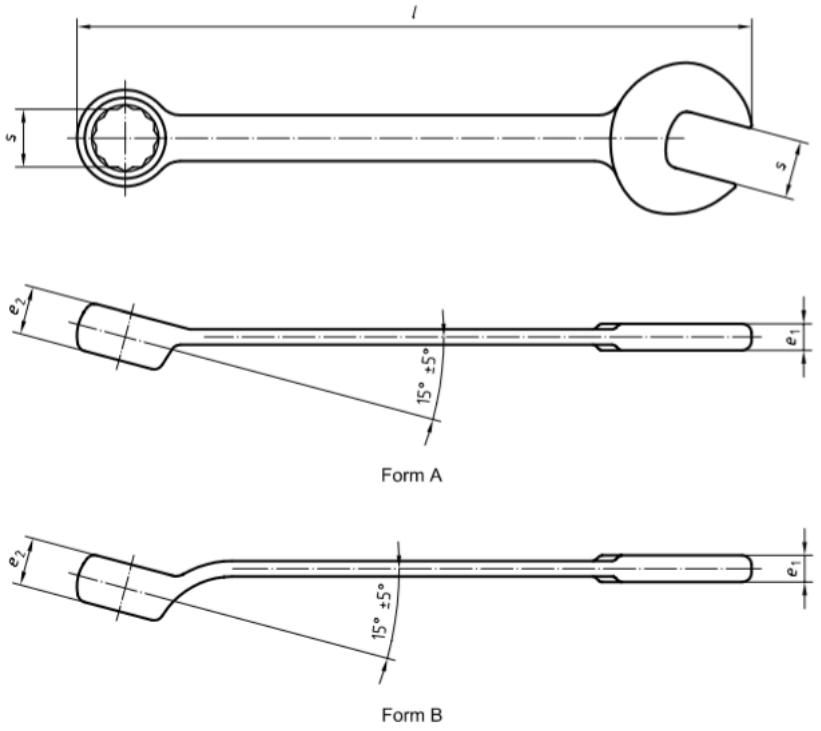


FIGURA 3.7: ISO 7738 Combinational wrench.

Opening *s*	Series "short"		Series "medium"		Series "long"	
	l_{max}	l_{max} (non-preferred)	l_{min}	l_{max}	l_{min} (non-preferred)	l_{min}
10	109	115	110	155	135	156
11	114	120	115	160	140	161
12	124	135	125	175	155	176
13	134	145	135	190	165	191
14	144	150	145	200	175	201
15	149	155	150	210	185	211
16	159	170	160	225	205	226
17	169	180	170	240	215	241
18	179	190	180	250	230	251
19	184	200	185	260	230	261

FIGURA 3.8: ISO 7738 Overall Lengths.

Questi fattori hanno di fatto impedito la costruzione di un modello preciso sia tridimensionale sia bidimensionale delle chiavi. D'altra parte, la challenge presentava anche delle caratteristiche vantaggiose, in quanto gli oggetti da trovare erano disposti in una

porzione ben precisa di ambiente, ossia il pannello, con sfondo scuro uniforme. Si è quindi optato di testare l'object detector implementato da OpenCV[1], libreria già utilizzata ampiamente in altri ambiti della challenge.

3.2.1 OpenCV cascade LBP

OpenCV fornisce classi e metodi per creare e applicare un boosted cascade classifier, basato sulle LBP feature. Queste ultime garantiscono infatti un tempo di training notevolmente inferiore rispetto alle precedenti Haar feature, mantenendo ottime percentuali di successo. Il tool fornito da OpenCV è chiamato *opencv_traincascade*, scritto in C++, e sostituisce l'ormai deprecato *opencv_haartraining* aggiungendo il supporto alle LBP feature. La classe di riferimento è chiamata *CascadeClassifier*, inserita all'interno del modulo *objdetect*.

Per quanto riguarda la fase di creazione del classificatore, sono state utilizzate le seguenti funzioni:

- *opencv_createsamples*: crea un set di numerosi samples positivi a partire da immagini contenenti l'oggetto; applica una serie di rotazioni casuali, modifica l'intensità dell'immagine e la sovrappone a diverse immagini di background. Questa tecnica è chiamata *data augmentation*, e serve ad aumentare il numero di samples positivi;
- *opencv_traincascade*: allena il classificatore sul dataset fornito, cercando di raggiungere gli obiettivi fissati dall'utente (*minHitRate* e *maxFalseAlarmRate*).

Terminata la fase di training il classificatore viene salvato in un file *.xml* e può essere utilizzato come oggetto indipendente in sistemi informatici diversi da quello di creazione.

Per quanto riguarda la detection, e quindi l'effettivo utilizzo del classificatore, OpenCV fornisce due metodi che lavorano in sinergia:

- *detectMultiScale*: applica il classificatore sull'immagine data in input, scalandolo in differenti dimensioni; gli oggetti identificati vengono restituiti come lista di rettangoli.

[1] OpenCV: http://opencv.org

- *groupRectangles*: raggruppa le possibili detection in cluster in base alla vicinanza, scartando quelli con un numero di rettangoli inferiore alla threshold impostata. Restituisce anche il "peso" di ogni detection, ossia il numero di vicini nel cluster di appartenenza.

In aggiunta agli strumenti forniti da OpenCV, si è scelto di utilizzare anche il pacchetto creato da *Naotoshi Seo* [20], che fornisce una serie di utili script per la creazione dei samples a partire dalle immagini del database.

3.2.2 Detection delle chiavi

Dataset

Per l'identificazione delle chiavi combinate è stato necessario creare un database consistente, tenendo in considerazione numerose condizioni di luce e differenti forme di chiavi. Le acquisizioni sono state ottenute con la stereo camera *BumbleeBee 2* e le due *Grasshopper 3*, in ambiente indoor e in ambiente outdoor. In quest'ultimo caso sono state effettuate 15 registrazioni video da circa un minuto l'una ogni trenta minuti, così da coprire un numero consistente di condizioni di luce; sono state inoltre utilizzate chiavi combinate appartenenti a quattro set diversi. Si è deciso di creare un classificatore che riguardasse esclusivamente le teste delle chiavi, proprio per impedire che la parte più variabile, ossia lo stelo, influisse negativamente sulla detection. Inoltre, analizzando solo la testa delle chiavi si evitava il disturbo causato dai pioli, anch'essi riflettenti e di dimensione non definita. Un ulteriore aspetto che ha supportato questa scelta è il fatto che il classificatore stesso avrebbe isolato la zona di maggior interesse per le successive fasi della challenge: l'analisi della testa è infatti di estrema importanza per definire l'orientazione della chiave e il punto centrale dell'apertura, necessario per l'inserimento della chiave stessa nella valvola.

Per il **dataset positivo** si è proceduto a ritagliare un'area quadrata che facesse da bounding box alla testa della chiave, la più ridotta possibile. Ogni immagine ritagliata è stata ridimensionata a *100x100 pixel*, un giusto equilibrio per mantenere un livello di dettaglio sufficiente e tempi di training non eccessivamente elevati. Tramite il processo di *data augmentation* si sono così ottenute **580 immagini positive**, rappresentanti la testa della chiave (Figura 3.9).

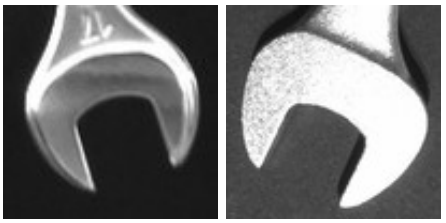


FIGURA 3.9: Positive Samples.

Per quanto riguarda il **dataset negativo**, sono state inserite immagini non contenenti teste delle chiavi, bensì oggetti o sfondi ritenuti probabili nel contesto della challenge: particolare attenzione si è quindi prestata sui pioli, sulla valvola e sul pannello, oltre ad immagini e forme generiche. OpenCV non impone vincoli sulle dimensioni (in pixel) delle immagini negative se non che siano superiori alle dimensioni del classificatore stesso, perciò nel dataset sono state inserite immagini di grandezza diversa. Il numero di **background samples** raccolti è di **5150** elementi.

La costruzione dei dataset ha richiesto tempo e vari esperimenti prima di raggiungere un risultato soddisfacente: inizialmente infatti, per ogni prova si sono raccolti i falsi positivi ed i falsi negativi del classificatore corrente, raccogliendo nuove acquisizioni che li comprendessero per incrementare il dataset sia negativo sia positivo. Questa tecnica è chiamata *hard negative mining*[9], e permette di creare progressivamente classificatori sempre più performanti per il task richiesto. Una importante modifica ai dataset è avvenuta poi quando gli organizzatori della challenge hanno modificato il colore del pannello su cui sono appese le chiavi da bianco a nero. I test effettuati avevano infatti portato alla luce un problema che rendeva il classificatore di allora instabile: in condizioni di luce intensa le ombre delle chiavi sul nuovo pannello costituivano una forma scura su sfondo chiaro, molto simili alle immagini positive del dataset precedente (chiavi scure su sfondo chiaro). Il classificatore confondeva le ombre delle chiavi con le chiavi stesse (Figura 3.10), generando dei falsi positivi.

Si è scelto perciò di confrontare due classificatori creati con differenti dataset positivi: nel primo si sono mantenute le acquisizioni fatte in precedenza con chiavi su pannello chiaro, oltre alle nuove acquisizioni con il pannello nero; nel secondo dataset sono stati invece rimossi tutti i samples raffiguranti chiavi scure su sfondo chiaro, simili alle ombre, mantenendo solo l'inverso, ossia chiavi chiare su sfondo scuro, delle nuove acquisizioni. I risultati del confronto hanno confermato l'ipotesi iniziale, con il primo classificatore

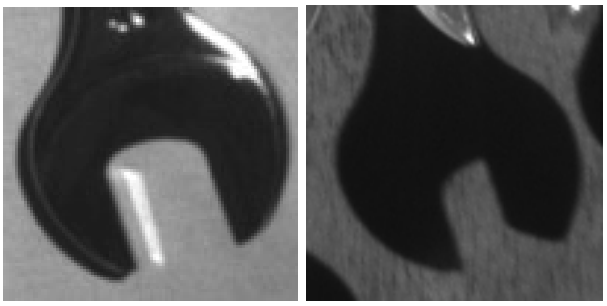


FIGURA 3.10: Similarità tra chiavi del dataset precedente (a sinistra) e ombre sul nuovo pannello (a destra).

che mostrava un eccessivo numero di falsi positivi, rappresentati dalle ombre, mentre il secondo classificatore testato sulle medesime immagini riusciva a non confondere le ombre come chiavi (Figura 3.11).

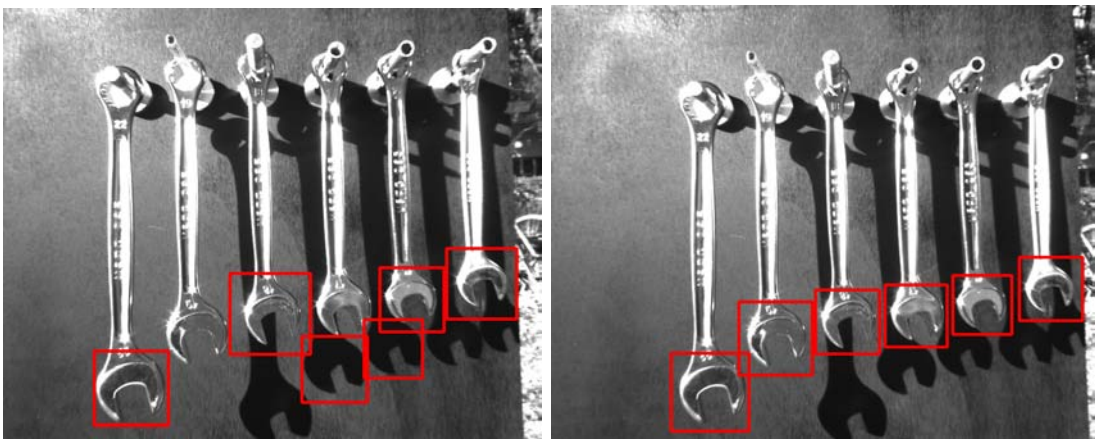


FIGURA 3.11: Detection con il dataset precedente (sx) e con il dataset aggiornato (dx).

Parametri del classificatore

Nella costruzione del classificatore sono stati testati vari parametri, fino ad ottenere i risultati desiderati. Nell'elenco seguente sono illustrati i valori impostati e la motivazione che ha portato alla loro scelta.

1. Script di Naotoshi Seo per la creazione dei *positive samples*:

   ```
   perl bin/createsamples.pl pos_wrench.txt neg_wrench.txt samplesWrench
   "opencv_createsamples -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5
   -maxidev 40 -w 32 -h 32 -num 2000"
   ```

- **pos_wrench.txt/neg_wrench.txt**: files di testo contenenti i nomi e i percorsi rispettivamente delle immagini positive e di quelle negative;
- **samplesWrench**: cartella in cui vengono salvati i samples positivi creati;
- **-maxxangle/maxyangle/maxzangle**: angolo massimo di rotazione dell'immagine iniziale rispetto all'asse x,y e z rispettivamente. Essendo le chiavi appese in verticale, si è scelto di impostare una rotazione massima di **0.5 radianti**, sufficiente a coprire errori di allineamento del braccio manipolatore, e quindi delle camere, rispetto al pannello;
- **-maxidev**: massima variazione di intensità dell'immagine iniziale. E' stato impostato il valore medio di **40**, in quanto già nelle acquisizioni si è tenuto conto di varie condizioni di luce;
- **-w/-h**: larghezza (width) e altezza(height) in pixel del classificatore. Come spiegato in precedenza, si è scelto di analizzare solo la testa delle chiavi, ottimamente racchiuse da un bounding box di forma quadrata. Il valore di **32 pixel** si è dimostrato sufficiente per la distanza di lavoro prevista e la risoluzione delle camere, garantendo tempi relativamente bassi di training;
- **-num**: numero di samples positivi generati sovrapponendo le immagini positive a quelle negative di background, tenendo conto dei parametri precedenti riguardo rotazione e intensità. E' stato scelto il valore di **2000** in modo da ottenere 3/4 samples per ogni immagine di partenza.

2. Script di Naotoshi Seo per l'unione (*merge*) dei vari samples in un unico vettore:

   ```
   python ./tools/mergevec.py -v samplesWrench/ -o samplesWrench.vec
   ```

   - **-v**: cartella in cui sono salvati i samples positivi creati;
   - **-o**: vettore, restituito in output, contenente tutti i samples positivi creati, come richiesto da OpenCV.

3. Funzione di OpenCV per il training del classificatore:

   ```
   opencv_traincascade -data classifierWrench -vec samplesWrench.vec -bg
   neg_wrench.txt -numStages 14 -minHitRate 0.998 -maxFalseAlarmRate 0.33
   -numPos 1650 -numNeg 5150 -featureType LBP -bt DAB -w 32 -h 32
   -precalcValBufSize 2048 -precalcIdxBufSize 2048
   ```

   - **-data**: cartella di output per il classificatore;

- **-vec**: vettore contenente tutti i samples positivi;
- **-bg**: file di testo relativo ai samples negativi;
- **-numStages**: numero di stage per il training del classificatore: è stato osservato che **14 stage** sono sufficienti (in base alle dimensioni del dataset) per raggiungere gli obiettivi di dei due parametri successivi;
- **-minHitRate**: minimo valore di hit rate (samples positivi correttamente riconosciuti) desiderato per ogni stage; il valore totale raggiunto dal classificatore può essere calcolato come $minHitRate^{numStages}$. E' stato scelto un minHitRate di **0.998**, in quanto nell'ambito della challenge è preferibile avere più falsi positivi che falsi negativi.
- **-maxFalseAlarmRate**: massimo valore di false alarm rate (samples erroneamente classificati come positivi) desiderato per ogni stage; è stato scelto un maxFalseAlarmRate di **0.33**. Il valore totale raggiunto dal classificatore può essere calcolato come $maxFalseAlarmRate^{numStages}$;
- **-numPos**: numero di samples positivi da utilizzare in ogni stage; un valore corretto varia dall'80% al 90% di tutti i samples positivi, quindi è stato imposto un numPos di **1650 samples**.
- **-numNeg**: numero di samples negativi da utilizzare in ogni stage;
- **-featureType**: il tipo di feature scelte, che per OpenCV possono essere Haar o LBP; come spiegato precedentemente sono state utilizzate le **LBP** feature, che permettono un training diverse volte più veloce;
- **-bt**: tecnica di boosting da utilizzare; è stata mantenuta la tecnica **Gentle Adaboost**, assegnata di default da OpenCV in quanto nella maggior parte delle situazioni si rivela più performante delle altre;
- **-precalcValBufSize/-precalcIdxBufSize**: dimensione del buffer da allocare per il processo di training, in Mb.

Il tempo di training con questi parametri è stato di circa **1 giorno e 8 ore**.

Detection e post processing

Creato il classificatore si è potuto passare alla fase di detection e di post processing, che ha permesso di perfezionare i risultati riducendo la possibilità di false detection.

Infatti le caratteristiche della challenge hanno consentito di imporre alcune condizioni stringenti: una prima modifica è stata ad esempio di mantenere solo le sei detection con confidenza maggiore, dato che le chiavi appese erano sempre in numero di sei. Come valore di confidenza è stato preso il peso restituito dal metodo *groupRectangles*, dato dal numero di detection presenti nel cluster in esame (Figura 3.12).

FIGURA 3.12: Cluster di detection prima di applicare *groupRectangles*.

Un altro vincolo che si è potuto imporre riguarda la vicinanza delle detection: infatti da specifiche le chiavi sono appese su pioli alla stessa altezza e a distanza di 5 cm rispetto ai loro centri. Questo ha permesso di scartare eventuali detection eccessivamente lontane dalle altre, e di mantenere solo quelle adiacenti.

Un'ulteriore condizione riguarda le dimensioni delle chiavi: conoscendo la distanza di lavoro si è potuto imporre un range di dimensioni bidimensionali nel metodo della detection, così da cercare solo oggetti di quella misura.

3.2.3 Detection della valvola

La costruzione di un classificatore per il riconoscimento della valvola ha seguito principalmente il procedimento già descritto per le chiavi, con alcune differenze.

La valvola presenta un aspetto tridimensionale con maggiore profondità dovuta allo stelo, e questo comporta variazioni notevoli dell'immagine bidimensionale (Figura 3.13).

Tuttavia nel caso della challenge la valvola è l'unico oggetto che si distacca dallo sfondo uniforme nella zona circostante, facilitando la detection. Un altro aspetto da considerare è l'ombra dello stelo sulla valvola stessa, e le variazioni di riflessi su una superficie maggiore rispetto a quella della testa delle chiavi. Inoltre lo stelo quadrato può essere ruotato in qualsiasi angolazione, modificando notevolmente l'aspetto totale dell'oggetto. Ciò che permette di limitare la complessità della classificazione è la conoscenza della forma della valvola, comprese le sue dimensioni esatte.

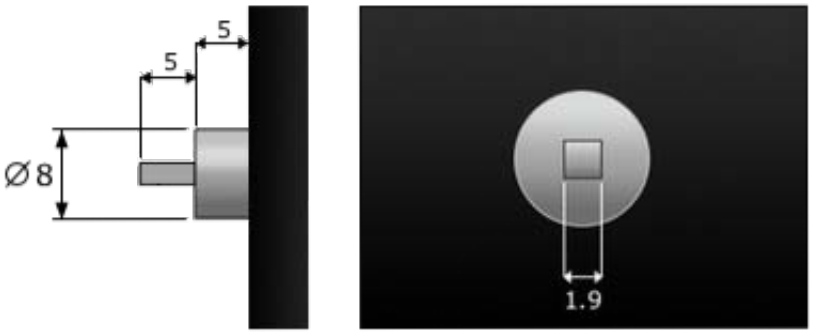


FIGURA 3.13: Vista laterale e frontale della valvola (dimensioni in cm).

Dataset

Per la creazione del dataset sono state acquisite sei registrazioni con le telecamere Grasshopper 3 in ambiente indoor, con luce uniforme, e in ambiente outdoor, con esposizione diretta alla luce del sole. Durante le acquisizioni il pannello è stato mosso e ruotato in varie direzioni così come lo stelo della valvola, così da coprire più casi possibile.

Per quanto riguarda il **dataset positivo**, si è ripetuto il processo di creazione dei samples e di *data augmentation* utilizzato per le chiavi, ritagliando un bounding box attorno alla valvola, applicando le trasformazioni descritte in precedenza e ridimensionando le immagini a *100x100 pixel*. Si sono così ottenute **315 immagini positive**.

Per il **dataset negativo** si è partiti da quello delle chiavi, eliminando successivamente immagini contenenti la valvola e aggiungendo i positive samples delle chiavi. Si sono così ottenuti **5245 background samples**.

Dopo aver creato il dataset per il classificatore della valvola si è proceduto ad irrobustire ulteriormente quello delle chiavi, andando ad aggiungere al dataset negativo le immagini positive della valvola, così da evitare false detection.

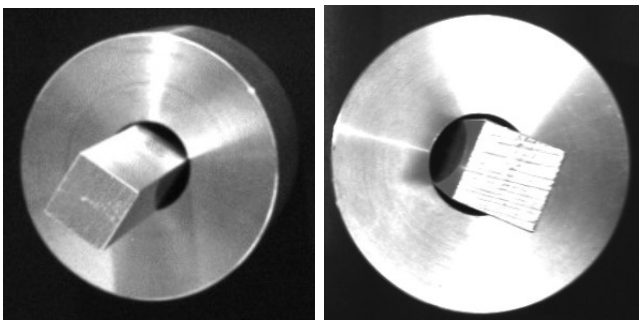


FIGURA 3.14: Positive Samples.

Parametri del classificatore

I parametri utilizzati per allenare il classificatore della valvola differiscono da quelli per il classificatore delle chiavi nei casi seguenti:

1. Script di Naotoshi Seo per la creazione dei *positive samples*:

   ```
   perl bin/createsamples.pl pos_valve.txt neg_valve.txt samplesValve
   "opencv_createsamples -maxxangle 1 -maxyangle 1 -maxzangle 1
   -maxidev 50 -w 32 -h 32 -num 2000"
   ```

   - **pos_valve.txt/neg_valve.txt**: file di testo contenenti i nomi e i percorsi rispettivamente delle immagini positive e di quelle negative;
   - **samplesValve**: cartella in cui vengono salvati i samples positivi creati;
   - **-maxxangle/maxyangle/maxzangle**: angolo massimo di rotazione dell'immagine iniziale rispetto all'asse x,y e z rispettivamente. Essendo la valvola circolare, si è scelto di impostare una rotazione massima di **1 radiante**;
   - **-maxidev**: massima variazione di intensità dell'immagine iniziale. E' stato impostato il valore medio di **50**, per tenere conto di varie condizioni di luce;
   - **-w/-h**: larghezza (width) e altezza(height) in pixel del classificatore. La valvola si presta ottimamente ad essere racchiusa da un bounding box di forma quadrata. Il valore di **32x32 pixel** si è dimostrato sufficiente per la distanza di lavoro prevista e la risoluzione delle telecamere, garantendo tempi relativamente bassi di training;
   - **-num**: numero di samples positivi generati sovrapponendo le immagini positive a quelle negative di background, tenendo conto dei parametri precedenti

riguardo rotazione e intensità. E' stato scelto il valore di **2000** in modo da ottenere 6/7 samples per ogni immagine di partenza.

2. Script di Naotoshi Seo per l'unione (merge) dei vari samples creati in un unico vettore:

   ```
   python ./tools/mergevec.py -v samplesValve/ -o samplesValve.vec
   ```

   - **-v**: cartella in cui sono salvati i samples positivi creati;
   - **-o**: vettore, restituito in output, contenente tutti i samples positivi creati, come richiesto da OpenCV.

3. Funzione di OpenCV per il training del classificatore:

   ```
   opencv_traincascade -data classifierValve -vec samplesValve.vec
   -bg neg_valve.txt -numStages 12 -minHitRate 0.998
   -maxFalseAlarmRate 0.35 -numPos 1600 -numNeg 5200 -featureType LBP
   -bt DAB -w 32 -h 32 -precalcValBufSize 2048 -precalcIdxBufSize 2048
   ```

   - **-data**: cartella di output per il classificatore;
   - **-vec**: vettore contenente tutti i samples positivi;
   - **-bg**: file di testo relativo ai samples negativi;
   - **-numStages**: numero di stage per il training del classificatore: è stato osservato che **12 stage** sono sufficienti (in base alle dimensioni del dataset) per raggiungere gli obiettivi di dei due parametri successivi;
   - **-minHitRate**: minimo valore di hit rate (samples positivi correttamente riconosciuti) desiderato per ogni stage; il valore totale raggiunto dal classificatore può essere calcolato come $minHitRate^{numStages}$. E' stato scelto un minHitRate di **0.998**, in quanto nell'ambito della challenge è preferibile avere più falsi positivi che falsi negativi.
   - **-maxFalseAlarmRate**: massimo valore di false alarm rate (samples erroneamente classificati come positivi) desiderato per ogni stage; è stato scelto un maxFalseAlarmRate di **0.35**. Il valore totale raggiunto dal classificatore può essere calcolato come $maxFalseAlarmRate^{numStages}$;
   - **-numPos**: numero di samples positivi da utilizzare in ogni stage; un valore corretto varia dall'80% al 90% di tutti i samples positivi, quindi è stato imposto un numPos di **1600 samples**.

- **-numNeg**: numero di samples negativi da utilizzare in ogni stage;
- **-featureType**: il tipo di feature scelte, che per OpenCV possono essere Haar o LBP; come spiegato precedentemente sono state utilizzate le **LBP** feature, che permettono un training diverse volte più veloce;
- **-bt**: tecnica di boosting da utilizzare; è stata mantenuta la tecnica **Gentle Adaboost**, assegnata di default da OpenCV in quanto nella maggior parte delle situazioni si rivela più performante delle altre;
- **-precalcValBufSize/-precalcIdxBufSize**: dimensione del buffer da allocare per il processo di training, in Mb.

Il tempo di training con questi parametri è stato di circa **9 ore**.

Detection e post processing

Creato il classificatore, si è cercato di migliorare i risultati andando a studiare opportuni metodi di post processing.

Per lo scopo della challenge si è innanzitutto imposto un vincolo sul numero di detection, mantenendo solo la detection con confidenza maggiore data la presenza di una sola valvola. Come valore di confidenza è stato preso, come per le chiavi, il peso restituito dal metodo *groupRectangles*, equivalente al numero di detection presenti nel cluster selezionato.

Un ulteriore vincolo imposto ha riguardato le dimensioni della valvola: nel metodo di detection si è scelto un range di grandezze probabili, così da scartare oggetti di dimensioni errate.

3.2.4 Stima della lunghezza delle chiavi

La challenge MBZIRC richiedeva l'individuazione della chiave in grado di operare sulla valvola, le cui dimensioni erano note. Si sono valutati vari approcci per identificare il numero di ogni chiave appesa, compito reso difficoltoso per la scarsità di informazioni tecniche fornite. In particolare il dato più utile, non fornito se non poche settimane prima della gara, era quello riguardante la lunghezza delle chiavi, che avrebbe permesso di identificarle univocamente.

Inizialmente si è testato l'utilizzo di un **template matching**, andando a confrontare l'immagine di ogni chiave con dei template di riferimento, di cui si conosceva il numero corretto. La necessità di riscalare le immagini in base alla distanza e di precisione inferiore al millimetro ha portato a scartare questo metodo, vista anche la forma non ben definita delle chiavi.

Si è valutata poi la possibilità di sfruttare tecniche di **Optical Character Recognition** (OCR), andando a leggere direttamente il numero della chiave dalle immagini. L'approccio tuttavia presentava numerose complessità, vista la variabilità di posizione della numerazione, la presenza di altre scritte sullo stelo, e infine la non sicura presenza dei numeri nelle chiavi presenti in gara.

Il metodo che si è scelto di utilizzare si è rivelato, nella sua semplicità, estremamente robusto ed efficace (Risultati: 3.2.5). Invece di stimare per ogni chiave la sua numerazione si sono sfruttati alcuni vincoli e caratteristiche tecniche dati dalla challenge per ordinare le chiavi in base alla loro lunghezza relativa. Sapendo che esse sono appese a dei pioli alla medesima altezza, si è presa come riferimento della lunghezza di ogni chiave **la coordinata y del centro della detection** della testa. Infatti il centro del quadrato (bounding box della testa della chiave) restituito dal classificatore è un punto stabile, indipendente dalla grandezza del quadrato stesso. In base a questa coordinata si sono potute ordinare relativamente le chiavi dalla più corta alla più lunga (Figura 3.15).

Una volta ricevute le caratteristiche tecniche delle chiavi che sarebbero state usate in gara, grazie a questo approccio si è potuto anche assegnare con precisione il numero corrispondente ad ogni chiave. Infatti le chiavi disponibili avrebbero avuto numerazione 16 - 17 - 18 - 19 - 22 - 24, distribuite casualmente sui pioli. Per le dimensioni della valvola la chiave corretta era la 19, e con questo metodo la si è potuta identificare con precisione come la terza più lunga.

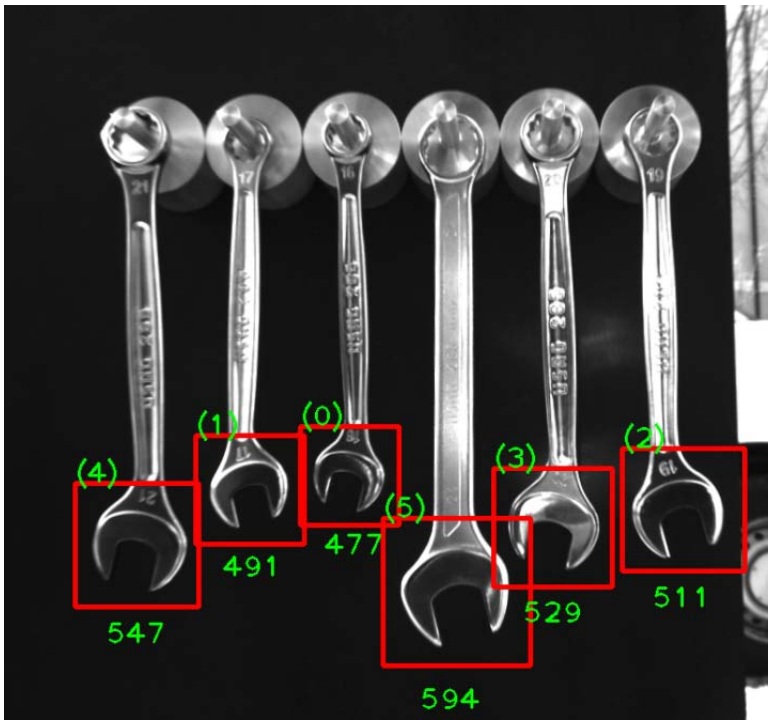


FIGURA 3.15: Ordinamento delle chiavi in base alla loro lunghezza relativa: il numero superiore indica l'ordinamento, dalla chiave più corta (0) a quella più lunga (5); il numero inferiore indica la coordinata Y, e quindi la misura relativa della lunghezza della chiave.

3.2.5 Risultati

Classificatore delle chiavi

Per testare le performance del classificatore delle chiavi è stato creato un nuovo dataset, dato che in quello utilizzato per il training i positive samples raffiguravano esclusivamente la testa della chiave. In questo nuovo dataset si sono invece potute inserire immagini di dimensioni diverse, raffiguranti anche l'intero pannello. Il dataset è stato così suddiviso:

- **225 immagini positive** del pannello, contenti una chiave (Figura 3.16);
- **75 immagini negative** del pannello, non contenenti nessuna chiave (Figura 3.17).

Si sono scelte chiavi appartenenti a due set diversi (Figura 3.18), uno dei quali non utilizzato per la fase di training. Le acquisizioni sono state effettuate in ambiente *indoor*, con illuminazione uniforme sul pannello, e *outdoor*, in condizioni di luce intensa. Sono state inoltre raccolte immagini da angolazioni e distanze differenti.

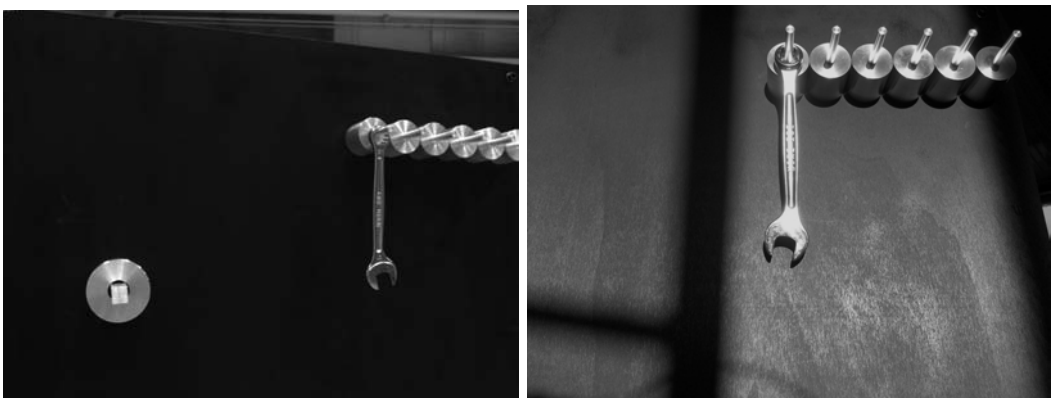


FIGURA 3.16: Immagini positive.

FIGURA 3.17: Immagine negativa.

FIGURA 3.18: Chiavi di due set diversi utilizzate per il testing.

Per la fase di **testing** è stato sviluppato un programma che legge progressivamente le immagini del dataset e applica il classificatore, localizzando la chiave se presente o notificando la sua assenza. Sono stati effettuati tre test, modificando di volta in volta il parametro di *minNeighbors*, che imposta una soglia minima di vicini per considerare l'oggetto una detection positiva. I risultati (Figure 3.19 e 3.20) sono stati classificati in:

- **True Positives (TP):** è stata identificata correttamente la chiave;
- **False Positives (FP):** è stato identificato erroneamente un altro oggetto come chiave;
- **False Negatives (FN):** non è stata trovata (erroneamente) nessuna chiave;
- **True Negatives (TN):** non è stata trovata (correttamente) nessuna chiave;

		CLASSIFICATORE	
		Trovata una chiave	Nessuna chiave trovata
REALTA'	Chiave presente	TP	FN
	Chiave non presente	FP	TN

FIGURA 3.19: Tabella di contingenza per il classificatore delle chiavi.

	Dataset Positivo			Dataset Negativo	
minNeighbors	FP	FN	TP	FP	TN
5	2,2 %	4,0 %	93,8 %	6,7 %	93,3 %
8	1,3 %	5,8 %	92,9 %	2,7 %	97,3 %
12	0,4 %	7,1 %	92,4 %	0,0 %	100,0 %

FIGURA 3.20: Risultati del classificatore delle chiavi sul dataset positivo e sul dataset negativo.

Analizzando i risultati ottenuti (Figure 3.21 e 3.22) si può notare come all'aumentare del parametro di soglia sul numero di vicini diminuiscano i falsi positivi (FP), e di conseguenza aumentino i veri negativi (TN) (Figure 3.23 e 3.24). D'altra parte anche il numero di veri positivi (TP) decresce all'aumentare del parametro.

Si sono inoltre calcolate altre misure per determinare le performance di un classificatore (Figura 3.25) [2]:

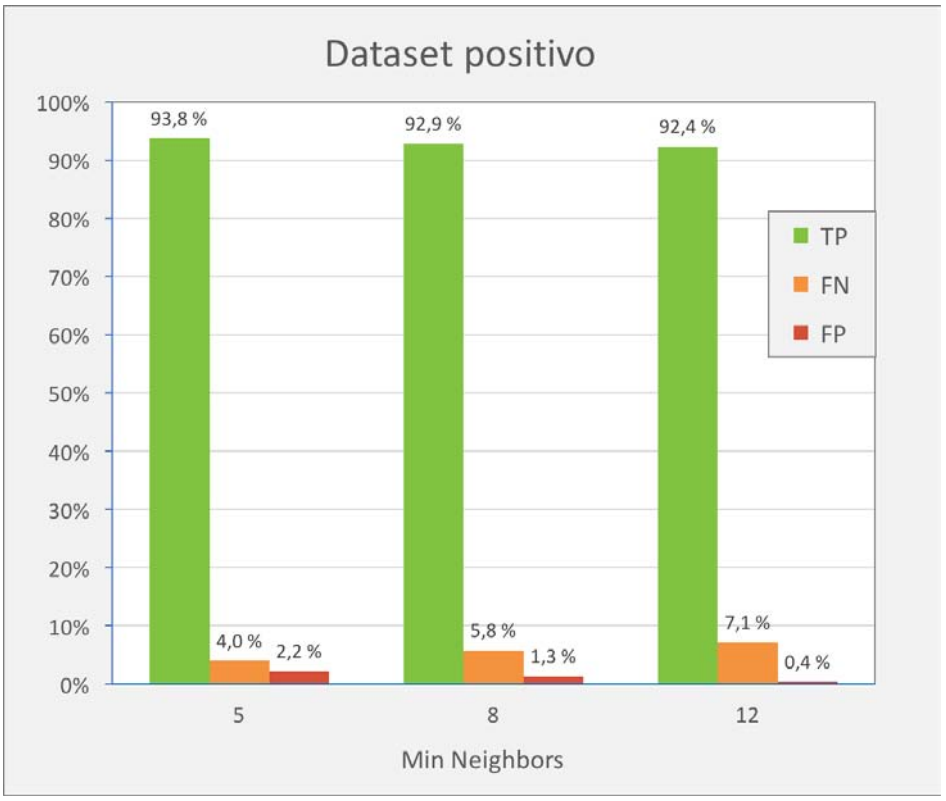


FIGURA 3.21: Risultati del classificatore delle chiavi sul dataset positivo.

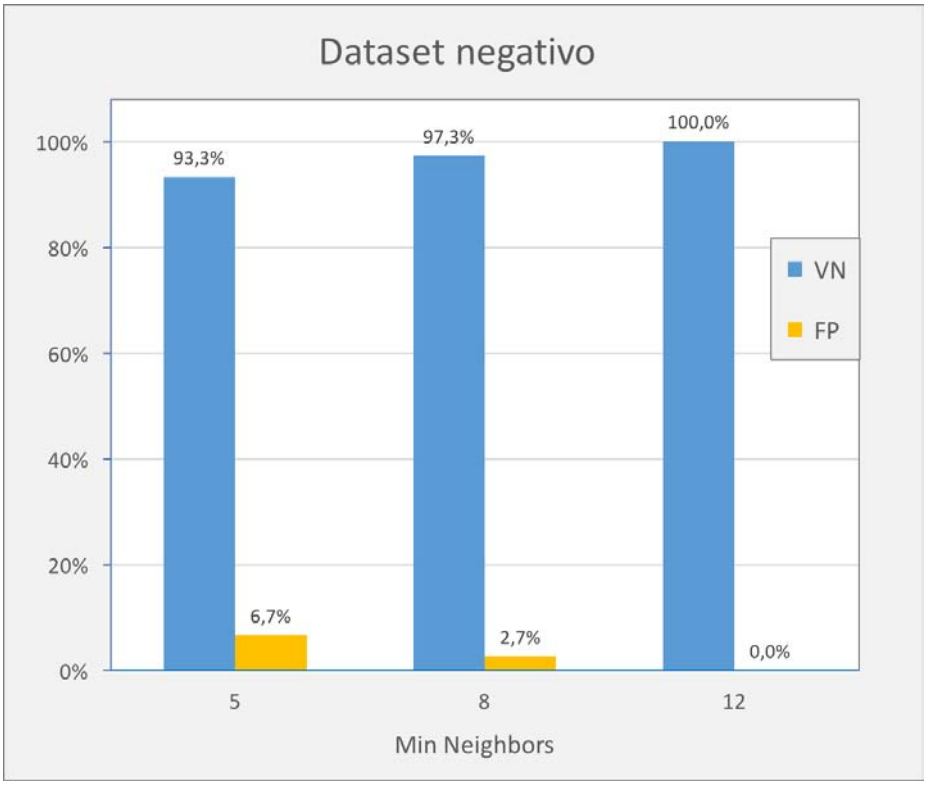


FIGURA 3.22: Risultati del classificatore delle chiavi sul dataset negativo.

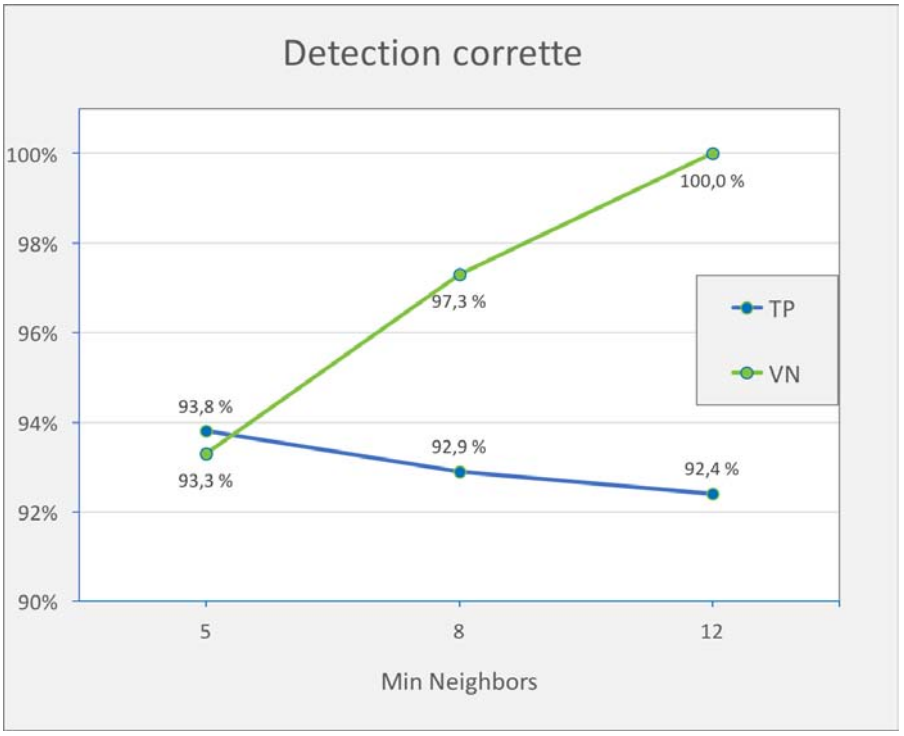


FIGURA 3.23: Detection corrette di chiavi (TP e TN).

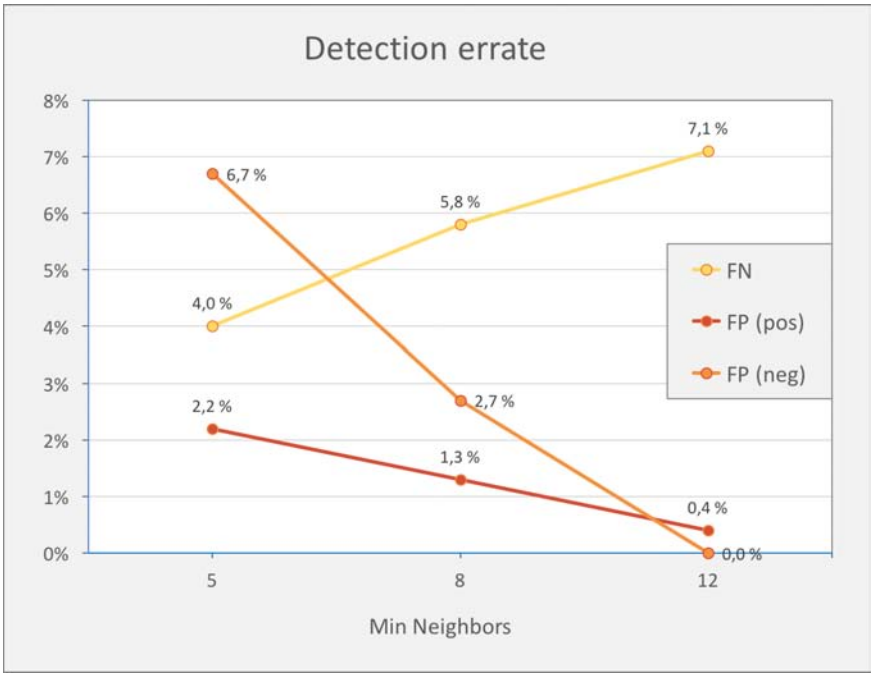


FIGURA 3.24: Detection errate di chiavi (FN e FP di entrambi i dataset).

- **Accuracy**: è il numero di detection corrette diviso il numero totale di previsioni.

$$\frac{(TP + TN)}{(TP + FP + FN + TN)}$$

- **Precision**: è il numero di previsioni positive corrette diviso il numero di previsioni appartenenti alla classe positiva. Indica l'*esattezza* e la *qualità* del classificatore. Un'alta precisione significa che l'algoritmo restituisce più risultati rilevanti che irrilevanti.

$$\frac{TP}{(TP + FP)}$$

- **Recall or Sensitivity**: è il numero di previsioni positive corrette diviso il numero di previsioni su dati appartenenti alla classe positiva nella realtà. Indica la *completezza* (in *quantità*) del classificatore. Un valore alto di recall significa che l'algoritmo restituisce la maggior parte dei risultati rilevanti.

$$\frac{TP}{(TP + FN)}$$

		minNeighbors		
		5	8	12
Precision	TP / (TP+FP)	95,4%	97,7%	99,5%
Accuracy	(TP+TN) / total	93,7%	94,0%	94,3%
Recall or Sensitivity	TP / (TP+FN)	95,9%	94,1%	92,9%

FIGURA 3.25: Misure per le performance del classificatore di chiavi.

Come si può osservare nella Figura 3.25, sia la precisione sia l'accuratezza aumentano con l'incremento del valore di minNeighbors. Viceversa, la misura di recall è migliore con valori bassi di minNeighbors, visto il minor numero di falsi negativi.

Data la presenza della fase di post processing, che permette efficacemente di riconoscere e scartare i falsi positivi, si è preferito avere un maggior numero di veri positivi, anche a discapito dell'aumentare dei falsi positivi. Si è quindi data maggiore importanza al valore della misura di recall. In questo modo si hanno maggiori possibilità di riconoscere tutte le chiavi, e le eventuali detection errate (falsi positivi) sono rigettate dal post processing. Si è quindi scelto come valore di *minNeighbors* **5 unità**.

Classificatore della valvola

Per testare il classificatore della valvola si è creato, per gli stessi motivi descritti in precedenza, un nuovo dataset con:

- **80 immagini positive** del pannello, contenti la valvola;
- **90 immagini negative** del pannello, non contenenti la valvola.

Le immagini sono state ottenute in ambiente *indoor*, con illuminazioni uniformi sul pannello, e *outdoor*, in condizioni di luce intensa e ombre. Le acquisizioni sono state effettuate da varie angolazioni e distanze differenti.

Il programma sviluppato per il **testing** ha le medesime funzioni di quello per le chiavi: legge progressivamente le immagini del dataset e applica il classificatore, individuando la valvola o notificando la sua assenza. Si sono svolti due test con valori del parametro di *minNeighbors* differenti; le performance ottenute (Figura 3.26) si suddividono in:

- **True Positives (TP):** è stata identificata correttamente la valvola (Figura 3.27);
- **False Negatives (FN):** non è stata trovata (erroneamente) nessuna valvola (Figura 3.28);
- **False Positives (FP):** è stato identificato erroneamente un altro oggetto come valvola (Figura 3.29);
- **True Negatives (TN):** non è stata trovata (correttamente) nessuna valvola (Figura 3.30).

		CLASSIFICATORE	
		Trovata una valvola	Nessuna valvola trovata
REALTA'	Valvola presente	TP	FN
	Valvola non presente	FP	TN

FIGURA 3.26: Tabella di contingenza del classificatore della valvola.

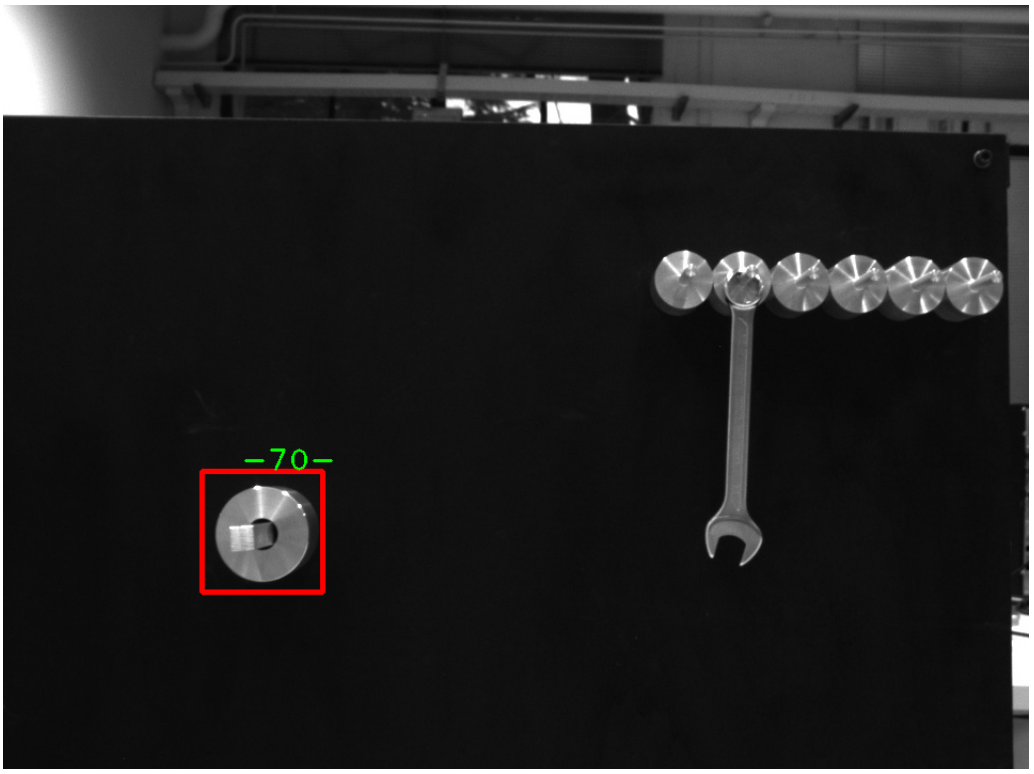


FIGURA 3.27: Esempio di TP: detection corretta della valvola. In verde è visualizzato il numero di *neighbors* della detection, che si è scelto di utilizzare come valore di confidenza.

FIGURA 3.28: Esempio di FN: valvola non trovata.

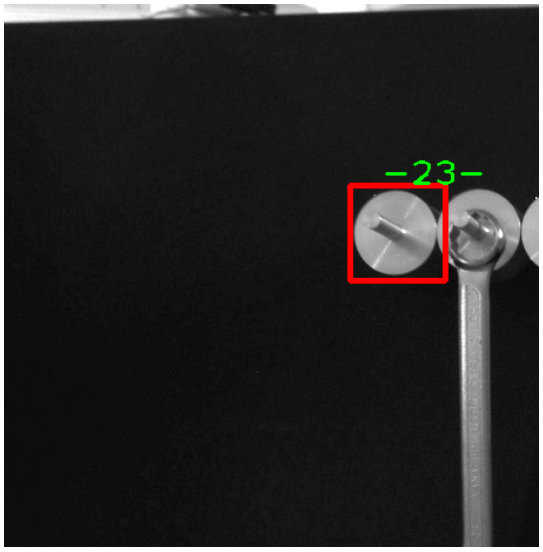


FIGURA 3.29: Esempio di FP: detection errata della valvola.

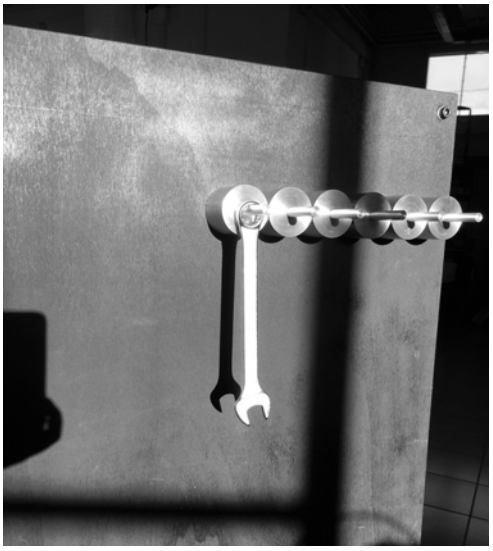


FIGURA 3.30: Esempio di TN: assenza della valvola.

I risultati conseguiti (Figure 3.31, 3.32 e 3.33) mostrano come abbassando il parametro di *minNeighbors* nel dataset positivo le detection corrette (TP) aumentino solo di poco, assieme tuttavia alle detection errate (FP) (Figure 3.34 e 3.35). D'altra parte nel dataset negativo al diminuire del valore di *minNeighbors* diminuisce anche il numero di TN e aumenta notevolmente quello dei FP.

	Dataset Positivo			Dataset Negativo	
minNeighbors	**FP**	**FN**	**TP**	**FP**	**TN**
3	3,7 %	2,5 %	93,8 %	15,6 %	84,4 %
8	1,2 %	6,3 %	92,5 %	4,4 %	95,6 %

FIGURA 3.31: Risultati del classificatore della valvola sul dataset positivo e negativo.

Si sono calcolate inoltre le misure per analizzare le performance di un classificatore descritte in precedenza: *precision*, *accuracy* e *recall* (Figura 3.36).

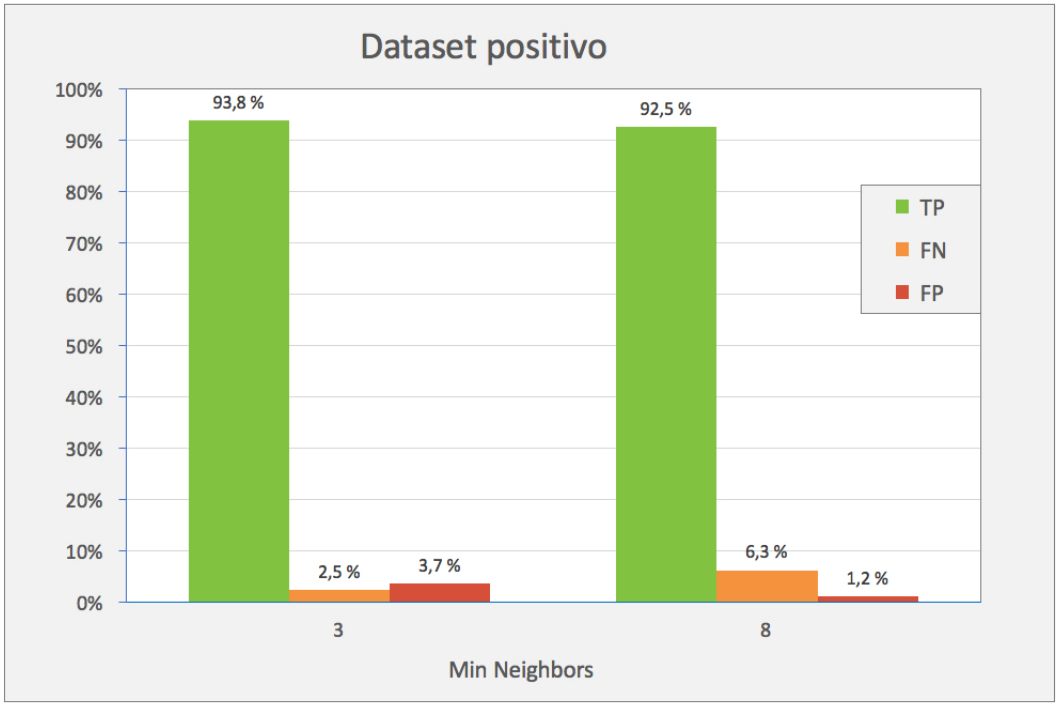


FIGURA 3.32: Risultati del classificatore sul dataset positivo.

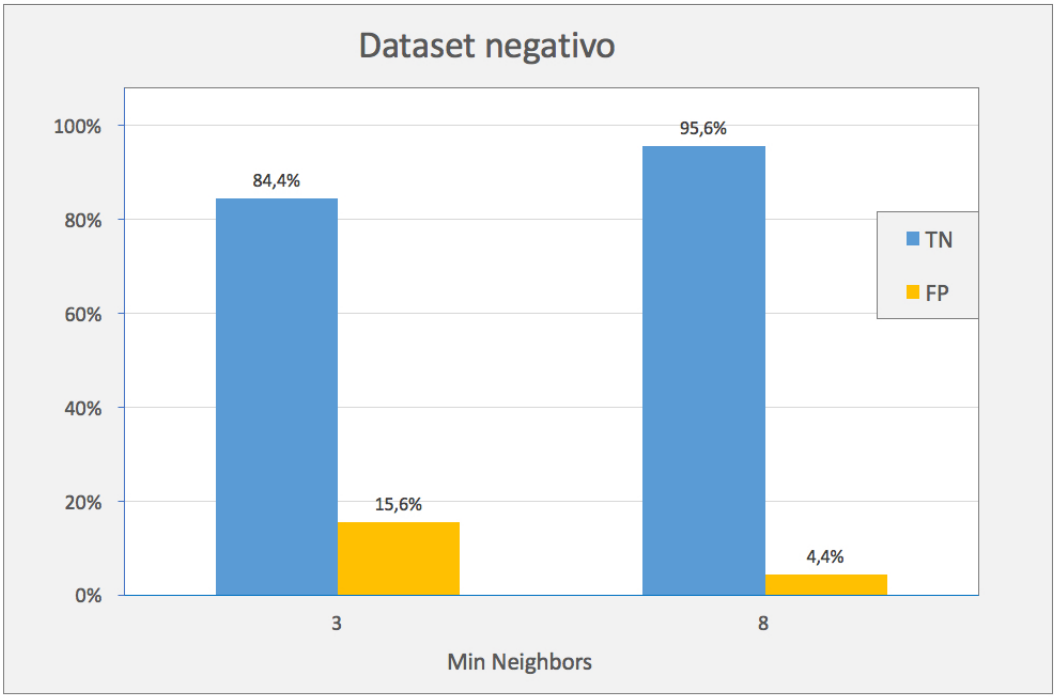


FIGURA 3.33: Risultati del classificatore sul dataset negativo.

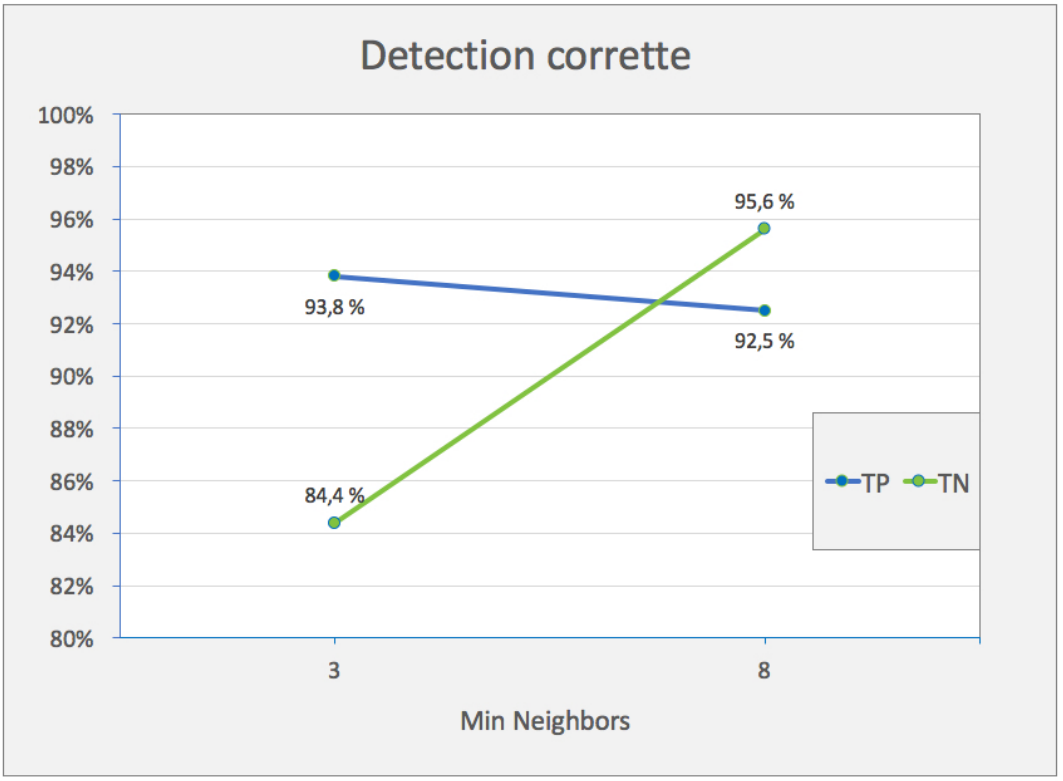


FIGURA 3.34: Detection corrette (TP e TN).

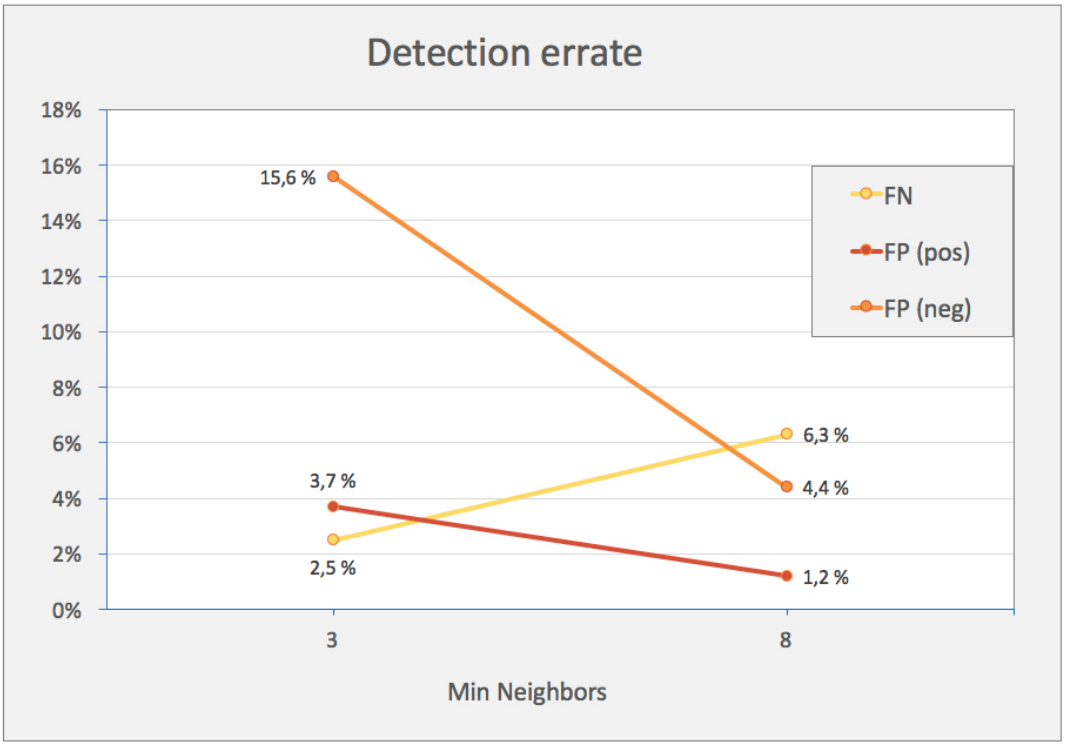


FIGURA 3.35: Detection errate (FN e FP di entrambi i dataset).

		minNeighbors	
		3	8
Precision	TP / (TP+FP)	81,5%	93,7%
Accuracy	(TP+TN) / total	88,8%	94,1%
Recall or Sensitivity	TP / (TP+FN)	97,4%	93,7%

FIGURA 3.36: Misure per le performance del classificatore della valvola.

Ispezionando i casi in cui il classificatore sbaglia nella detection, si sono potute elaborare delle osservazioni:

- La valvola non viene localizzata se la visuale del pannello è molto angolata (Figura 3.28): questo è coerente con il fatto che il training del classificatore è stato volutamente effettuato su acquisizioni frontali della valvola. Il motivo di questa scelta era la necessità di un classificatore robusto per il task specifico, che assicurava un posizionamento tale da garantire una vista non angolata della valvola.
- I falsi positivi sono quasi nella totalità dei casi pioli senza chiave appesa, di forma molto simile alla valvola (Figura 3.29). Nel caso della challenge tutti e sei i pioli presentano sempre una chiave appesa, situazione che presenta solo raramente dei falsi positivi.
- Per come è stato strutturato il task, il classificatore viene lanciato da una posizione in cui la valvola è sempre presente nelle immagini. Per questo motivo le performance ottenute con il dataset negativo sono state considerate meno rilevanti di quelle sul dataset positivo.

A seguito di questa analisi si è potuto impostare un valore di *minNeighbors* più **alto**, così da garantire un alto numero di detection corrette e un ridotto numero di errori.

3.2.6 Risultati in gara

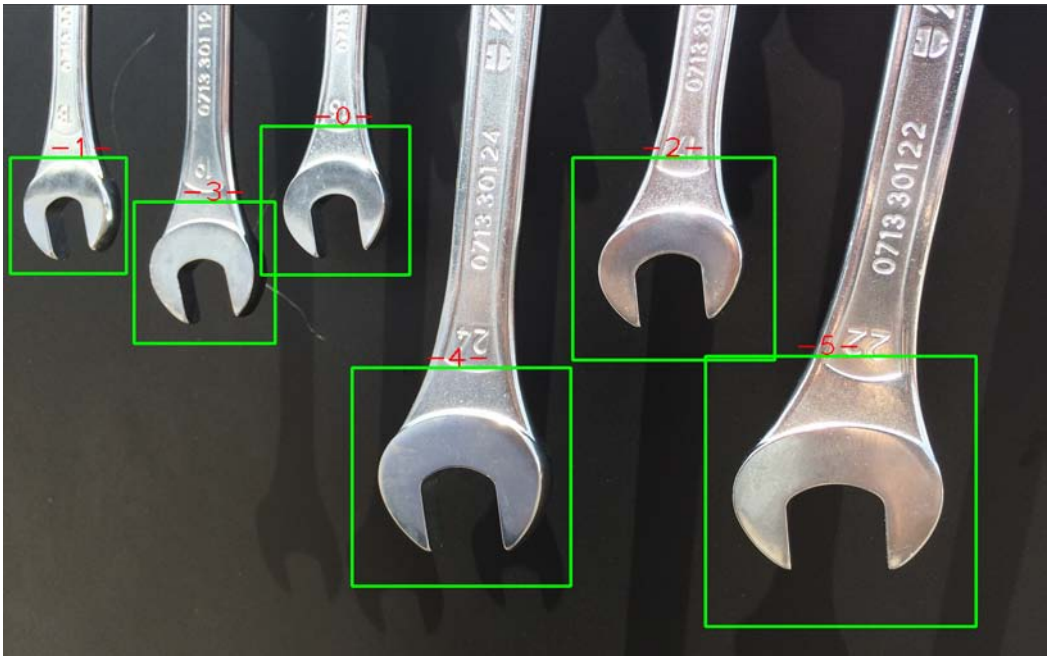
Classificatore delle chiavi

Il **classificatore delle chiavi** si è rivelato efficace anche nella challenge: nel corso delle prove si sono infatti riuscite a raccogliere alcune acquisizioni su cui si è verificato il suo corretto funzionamento. Il pannello e le chiavi sono stati fotografati da varie angolazioni e distanze, e con entrambe le rotazioni. Si sono così ottenute 50 immagini nell'arena di gara, con luce solare molto intensa.

Nonostante le chiavi in gara fossero di forma alquanto diversa dai set utilizzati per il training, il classificatore ha identificato correttamente **225/225 chiavi**, con una precisione del **100%** (Figura 3.37). Non si è verificato alcun falso positivo nè falso negativo, a confermare le ottime performance del classificatore specificatamente al task della challenge.

(a)

(b)



(c)

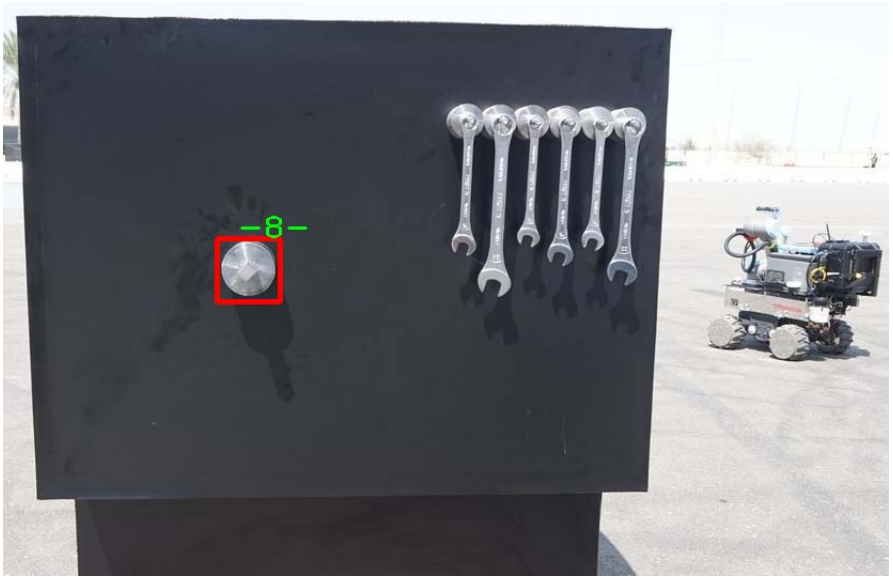
FIGURA 3.37: Localizzazione corretta di tutte le chiavi in gara. In rosso la posizione di ogni chiave in ordine di lunghezza crescente: 0 la più corta, 5 la più lunga.

Classificatore della valvola

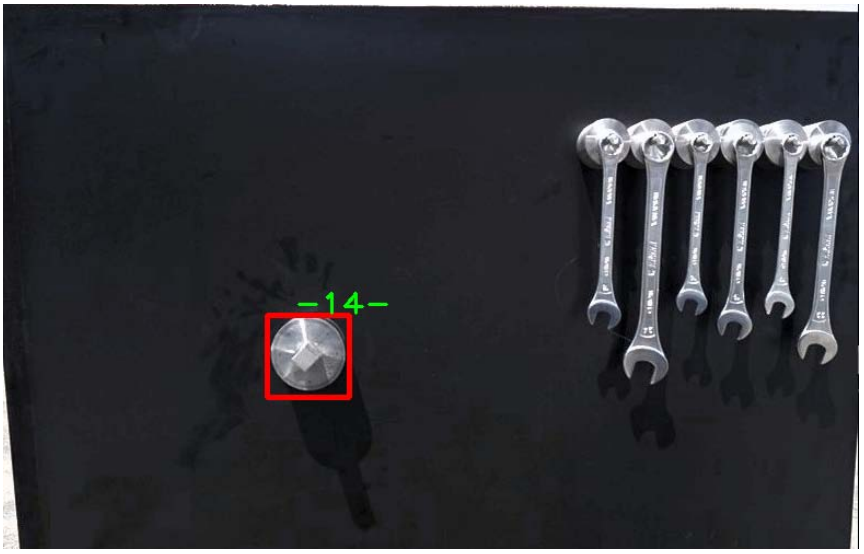
Per quanto riguarda il **classificatore della valvola**, è stato testato sulle poche acquisizioni che è stato possibile raccogliere in gara.

Su **40 immagini** contenti la valvola, il classificatore ha ottenuto i seguenti risultati (Figura 3.38):

- **90% detection corrette** (36/40 TP);
- **10% falsi positivi** (4/40 FP).



(a)



(b)

(c)

FIGURA 3.38: Detection corrette della valvola in gara.

Da notare come il classificatore sia riuscito ad ottenere ottimi risultati nonostante l'aspetto della valvola fosse notevolmente diverso dal previsto (Figura 3.39). In particolar modo il triangolo metallico indicante i gradi ha causato un peggioramento delle prestazioni, che sono tuttavia risultate sufficienti ad individuare la valvola nel 90% dei casi.

FIGURA 3.39: Valvola presente in gara.

Capitolo 4

Stima della posa 3D per presa ed inserimento

4.1 Point cloud e visualizzazione 3D

Con la crescita della potenza di calcolo dei processori ha acquisito sempre più importanza nella computer vision la rielaborazione dei dati tridimensionali. Richiedendo infatti una notevole capacità di memoria e calcoli computazionali complessi, solo in tempi recenti si è potuto lavorare con questi tipi di dati in tempo reale [23]. Un sensore che ha contribuito sensibilmente allo sviluppo del 3D nel campo della robotica è stato il Microsoft Kinect (Figura 4.1), creato per tutt'altri scopi. Da accessorio per la console di videogiochi Microsoft XBox 360, è diventato presto un sensore presente in ogni laboratorio di computer vision. La sua capacità di generare immagini tridimensionali in tempo reale per meno di $150 ha decretato un successo non previsto nemmeno dalla stessa Microsoft.

Figura 4.1: Microsoft Kinect XBox 360

Con l'avvento di questi nuovi ed economici sensori si è reso necessario sviluppare software in grado di processare e rielaborare i dati 3D. La libreria in assoluto più utilizzata e completa si chiama **PCL**[1] (*Point Cloud Library*): il concetto a cui rimanda il nome, *"nuvola di punti"*, è indicativo della natura dei dati di un'immagine tridimensionale. Una **point cloud** è più specificatamente una struttura dati costituita da una collezione di punti, rappresentanti le coordinate geometriche X, Y e Z di una superficie campionata (Figura 4.2).

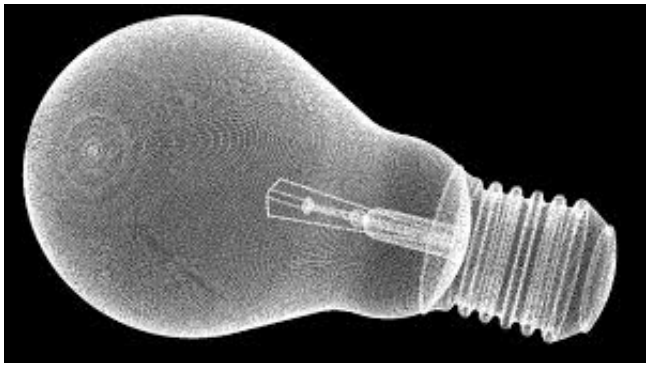


FIGURA 4.2: Point cloud 3D

Se l'informazione del colore è presente, la point cloud diventa una struttura 6D: XYZ-RGB (Figura 4.3).

FIGURA 4.3: Point cloud 4D

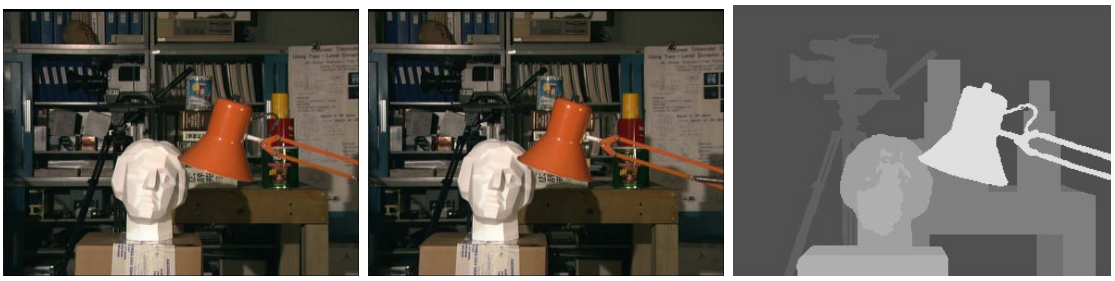
PCL fornisce numerosi algoritmi, frequentemente aggiornati, per la rielaborazione di dati 3D: filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration, ecc.

[1]Point Cloud Library: http://pointclouds.org

4.2 Stereo Vision

Un metodo alternativo a sensori come la Kinect per ottenere informazioni tridimensionali, ed eventualmente una point cloud, è rappresentato da un **sistema di visione stereo** costituito da due o più telecamere. Un sistema stereo copia essenzialmente la visione umana e di molti animali [13]: come noi abbiamo due occhi per vedere la stessa scena da angolazioni leggermente differenti, così in un sistema di visione stereo due telecamere sono posizionate ad una certa distanza l'una dall'altra (*baseline*). Con opportuni algoritmi di *matching* è possibile trovare corrispondenze tra punti chiave (*feature*) nelle varie immagini. A questo punto, conoscendo la posizione delle telecamere e le loro caratteristiche intrinseche, è possibile ottenere una *disparity map* della parte di immagine vista da entrambe le telecamere (*overlap area*). Una disparity map è un'immagine che permette di rappresentare efficacemente le distanze degli oggetti rispetto al punto di osservazione. L'informazione di disparity è ottenuta dalla differenza tra le coordinate orizzontali del punto in esame in entrambe le immagini (Figura 4.5). Da notare che il valore di disparity è inversamente proporzionale alla distanza effettiva dell'oggetto dalle telecamere nello spazio tridimensionale. Le Figure 4.4 e 4.5 (c) mostrano due esempi di disparity map: i colori più chiari indicano valori di disparity maggiori, quindi oggetti più vicini alle telecamere; viceversa valori di disparity minori sono rappresentati con colori più scuri, ed indicano una maggiore distanza dalle telecamere.

FIGURA 4.4: Esempio di disparity map.



(a) (b) (c)

FIGURA 4.5: Immagine sinistra (a) e destra (b) da cui si ricava la disparity map (c).

Nella pratica, per ottenere una visione stereo con due telecamere sono necessarie quattro fasi:

1. *Undistortion*: rimozione della distorsione radiale e tangenziale delle lenti (Figura 4.6 a);
2. *Rectification*: aggiustamento a seconda degli angoli e delle distanze tra le telecamere; l'output è costituito da immagini allineate orizzontalmente (row-aligned) e rettificate (Figura 4.6 b).
3. *Correspondence*: ricerca delle stesse feature in entrambe le immagini; l'output è una disparity map;
4. *Reprojection*: trasformazione dei valori di disparity in distanze tramite triangolazione, se a conoscenza delle posizioni geometriche delle telecamere; l'output è una depth map;

Nella Figura 4.7 è possibile osservare il caso semplificato e ottimale di un sistema stereo senza distorsioni, con piani immagini coplanari e row-aligned, assi ottici paralleli, e medesima distanza focale *f*. Assumendo di trovare le corrispondenze del punto spaziale P nei due piani immagini, in posizioni con coordinate orizzontali x^l e x^r, la disparity è data semplicemente da $d = x^l - x^r$. A questo punto, la **profondità Z** si può facilmente calcolare usando le proprietà geometriche dei triangoli simili (da qui il nome triangolazione):

$$Z = \frac{fB}{x^l - x^r}$$

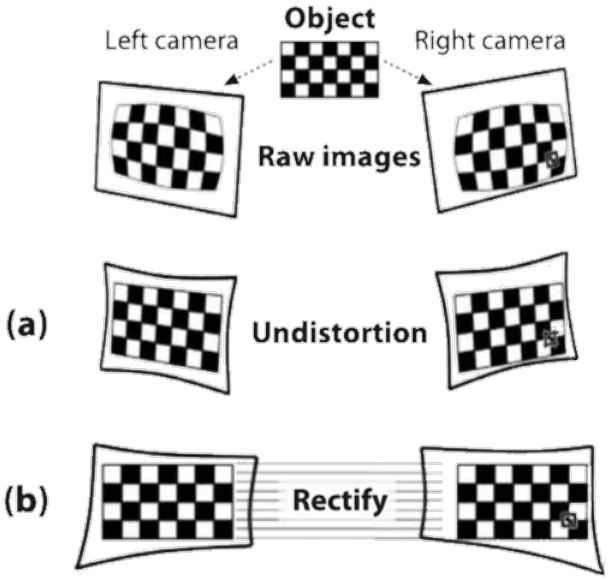


FIGURA 4.6: Processo di calibrazione delle immagini stereo

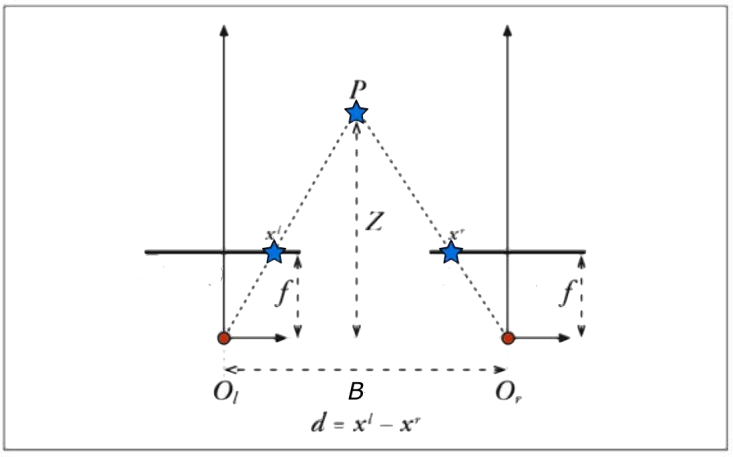


FIGURA 4.7: Caso semplificato e ottimale di sistema stereo

4.3 Pacchetto tf e coordinate frame

In un sistema robotico, di fondamentale importanza sono le posizioni e l'orientazione delle sue parti nel tempo, chiamate *coordinate frame*: ad esempio c'è il frame del mondo, il frame della base, il frame del gripper ecc (Figura 4.8). Per tenere traccia di queste informazioni si è scelto di utilizzare il **pacchetto ROS tf**[2]. Grazie a questo pacchetto è possibile organizzare e gestire i frame di coordinate in una *struttura ad albero*. Permette inoltre di creare facilmente nuovi frame, statici o dinamici, e si occupa di calcolare le *trasformazioni* per passare da un sistema di riferimento all'altro.

[2]Pacchetto ROS tf: http://wiki.ros.org/tf

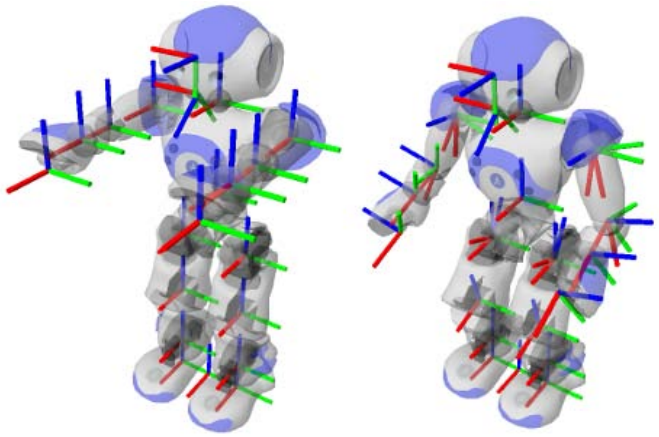


FIGURA 4.8: tf delle varie parti che compongono il robot Nao.

Il pacchetto tf può operare in un sistema distribuito: le informazioni sui frame di coordinate del robot sono disponibili ad ogni componente ROS, in tutti i computer del sistema. Non c'è infatti nessun server centrale che memorizza le informazioni sulle trasformazioni.

Le tf sono state di importanza notevole in questo lavoro di tesi, in quanto sono state usate per tener traccia della posizione e dell'orientazione dei vari oggetti, come le chiavi e la valvola, nel tempo.

4.4 Approccio per MBZIRC

Per quanto riguarda la challenge MBZIRC, il sistema di visione stereo era costituito da due telecamere *Grasshopper 3*, posizionate al di sopra del gripper (Figura 4.9).

FIGURA 4.9: Sistema di visione stereo del RUR53 con due Grasshopper 3.

I dati tridimensionali sono stati ottenuti tramite il pacchetto ROS `stereo_image_proc`[3] descritto nel paragrafo 4.4.1, che ricevendo in ingresso le immagini *raw* delle telecamere restituisce come output una *point cloud*. L'elaborazione e l'analisi della point cloud hanno permesso in particolare di eseguire le attività di grasping della chiave e di inserimento della stessa nella valvola (Figura 4.10). Analizzando le immagini 2D e passando successivamente al 3D si sono infatti posizionate le `tf` in punti strategici, opportunamente orientate. Si è potuto così comandare il robot in modo da allineare la `tf` dell'end-effector, il gripper, alla `tf` creata al centro della chiave scelta. Presa la stessa, si è spostato il braccio in modo da posizionarsi, con un opportuno offset di distanza, davanti alla `tf` della valvola. Infine per l'inserimento si sono sovrapposte la `tf` creata al centro della testa della chiave con la `tf` sullo stelo della valvola, entrambe orientate in base all'inclinazione dei due oggetti. Per definire l'orientazione delle chiavi e dello stelo della valvola si è utilizzata la *trasformata di Hough*, in grado di rilevare forme geometriche prestabilite in un'immagine.

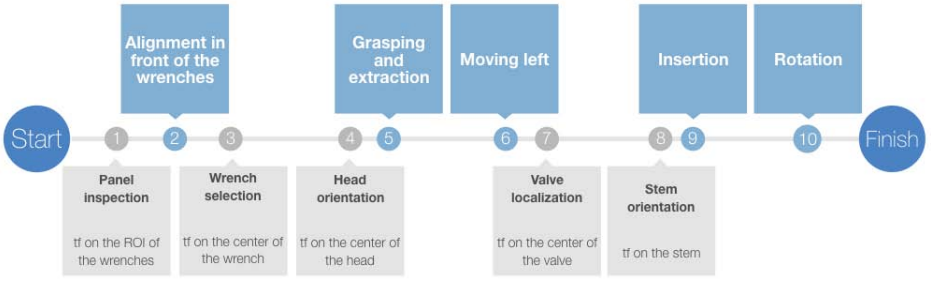


FIGURA 4.10: Operazioni per la presa della chiave e l'inserzione in valvola: in azzurro i movimenti del robot, in grigio i task di visione.

4.4.1 Metodi e pacchetti utilizzati

Pacchetto `stereo_image_proc`

Questo pacchetto ROS si occupa di gestire il sistema di visione stereo, facendo da intermediario tra i driver delle telecamere e i nodi di visione.

Nello schema di Figura 4.11 si può osservare come `stereo_image_proc` riceva in input le immagini raw e le informazioni sui parametri delle telecamere stesse (`camera_info`).

[3]ROS stereo_image_proc: http://wiki.ros.org/stereo_image_proc

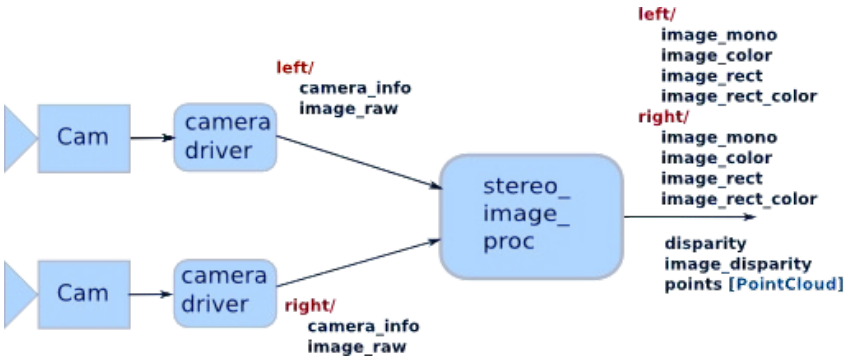


FIGURA 4.11: Pacchetto ROS `stereo_image_proc`.

Effettuati i processi di rimozione della distorsione e rettificazione, vengono restituite in output le immagini rettificate, mono, e a colori. `stereo_image_proc` si occupa inoltre di creare e pubblicare una *disparity image* e una point cloud: quest'ultima viene generata nel frame della telecamera sinistra (Figura 4.12).

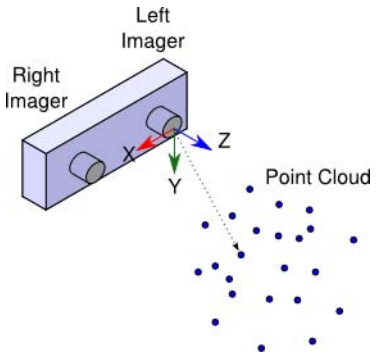


FIGURA 4.12: Posizionamento e orientazione del frame sulla telecamera sinistra.

Algoritmo `stereoBM`

Per generare le informazioni 3D il pacchetto `stereo_image_proc` utilizza la classe OpenCV `StereoBM`[4]. Questa sfrutta l'algoritmo per calcolare la corrispondenza stereo chiamato **Block Matching**[14]. Il nome deriva dal fatto che le immagini, in questo caso sinistra e destra, vengono suddivise in piccole regioni chiamate *blocchi* (block)[13]. Il matching viene quindi effettuato a blocchi, e per ogni blocco nell'immagine sinistra si cerca il match con il blocco più vicino nell'immagine destra. Essendo le immagini state

[4]Class StereoBM: http://docs.opencv.org/java/2.4.9/org/opencv/calib3d/StereoBM.html

rettificate e row-aligned, basterà cercare lungo linee orizzontali (Figura 4.13). La funzione di similarità per il matching dei blocchi è chiamata *Sum of Absolute Differences* (SAD).

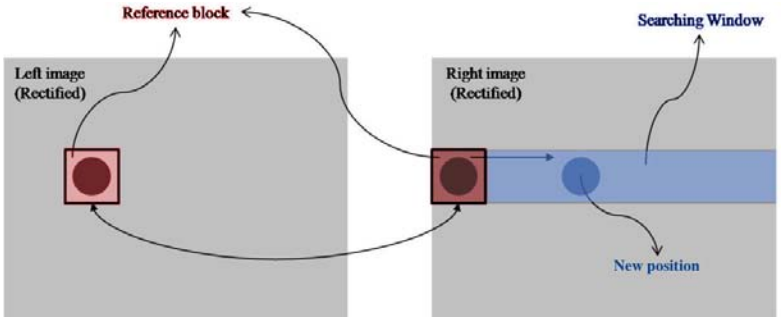


FIGURA 4.13: Funzionamento dell'algoritmo di stereo block matching.

Metodo `projectPixelTo3dRay`

Il metodo `image_geometry::PinholeCameraModel::projectPixelTo3dRay`[5] permette di passare da un punto 2D del piano immagine ad un punto 3D, avendo a disposizione le informazioni delle telecamere stereo (*camera_info*). Più nello specifico, questo metodo proietta un pixel rettificato su un raggio di punti 3D, restituendo un vettore nel coordinate frame della telecamera diretto verso il punto sul piano immagine (Figura 4.14).

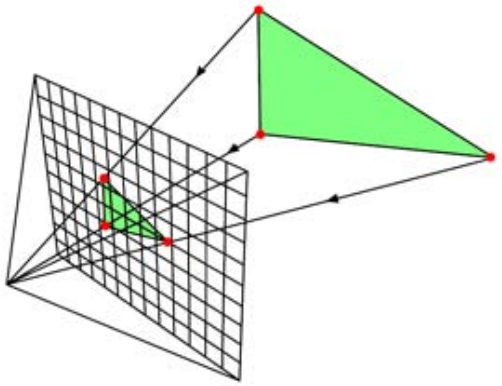


FIGURA 4.14: Proiezione di un punto sul piano immagine in un raggio 3D che collega la telecamera al pixel stesso.

[5]ProjectPixelTo3dRay: http://docs.ros.org/indigo/api/image_geometry/html/c++/classimage__geometry_1_1PinholeC

Metodo `SACSegmentation`

Il metodo SACSegmentation[6], della libreria PCL, permette di effettuare l'omonima *segmentazione Sample Consensus* (SAC). La segmentazione è un insieme di tecniche che hanno lo scopo di partizionare un'immagine in regioni omogenee. Ad esempio, è spesso utilizzata per identificare e isolare il pavimento o il terreno (*ground plane*). Quando si conosce già il modello geometrico da segmentare (piani, sfere, linee..), è spesso usato l'algoritmo randomized RANSAC (Random Sample Consensus)[15] (Figura 4.15).

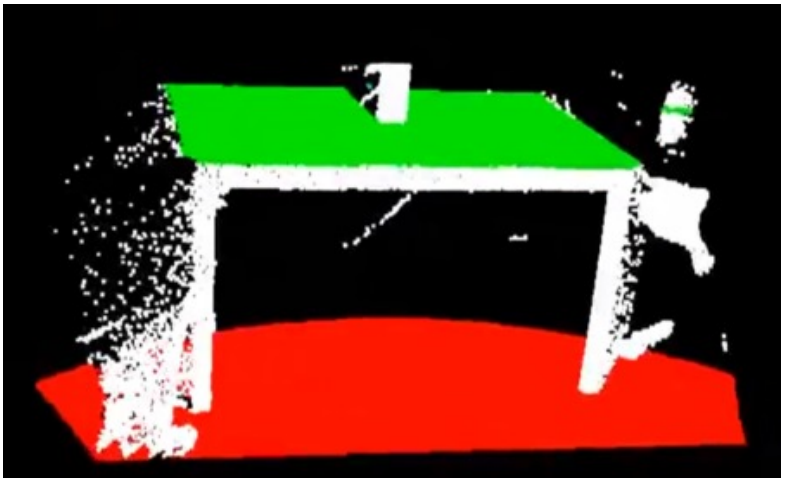


FIGURA 4.15: Segmentazione del pavimento e del piano del tavolo con RANSAC.

Trasformata di Hough

La **trasformata di Hough** è una tecnica che viene usata nella computer vision per identificare degli oggetti con una particolare forma in un'immagine[34]. Poiché è necessaria la descrizione parametrica della forma, la trasformata di Hough è prevalentemente applicata per la detection di *curve regolari* come linee e cerchi.

OpenCV implementa vari metodi che sfruttano la trasformata di Hough. Si prenda come esempio l'applicazione più utilizzata, la *Hough Line Transform*[22]. Una retta espressa in coordinate polari ha la seguente forma (Figura 4.16):

$$r = x \cos\theta + y \sin\theta$$

[6] pcl::SACSegmentation: http://docs.pointclouds.org/1.7.0/classpcl_1_1_s_a_c_segmentation.html

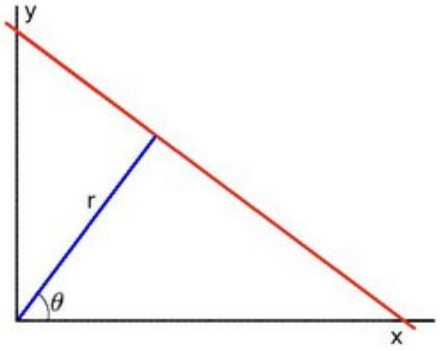


FIGURA 4.16: Retta in un sistema di coordinate polari.

Per un qualsiasi punto (x_0, y_0), ogni coppia (r_θ, θ) appartiene alla famiglia di linee che attraversano il punto. Se ognuna di queste viene mappata in un piano polare (r, θ) si ottengono delle sinusoidi. Visti in questo piano, punti che appartengono alla stessa linea nel piano cartesiano originale risaltano in quanto le loro curve si intersecano in un punto comune (Figura 4.17).

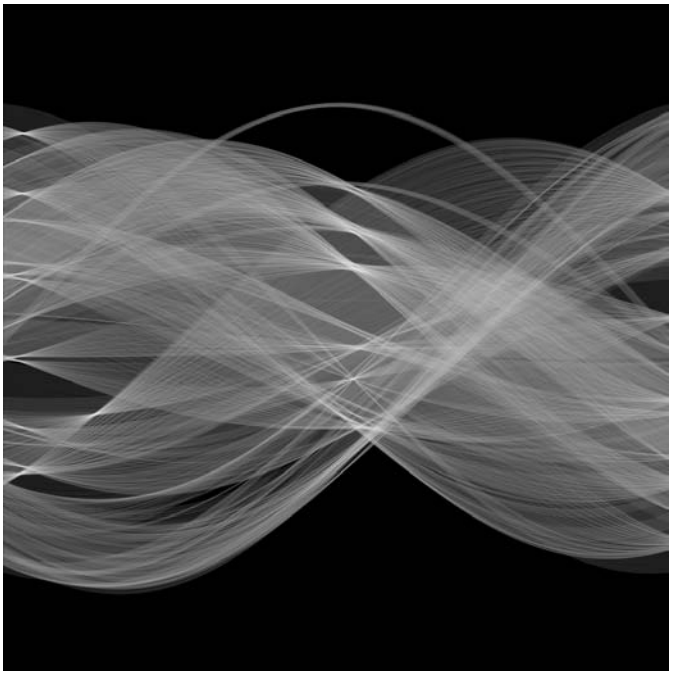


FIGURA 4.17: Rappresentazione delle sinusoidi della trasformata di Hough.

In conclusione, una linea viene rilevata con la trasformata di Hough cercando il numero di intersezioni tra le curve. Maggiori curve si intersecano, più punti avrà la linea rappresentata dal punto di intersezione. Si può quindi impostare un valore di soglia (*threshold*) che definisce il numero minimo di intersezioni per trovare una linea.

4.4.2 Ispezione del pannello e tf della ROI delle chiavi

Il primo task di visione della gara riguardava l'**ispezione del pannello**. Infatti il processo di localizzazione e successivo docking non teneva in considerazione il lato del pannello a cui ci si era affiancati, rendendo necessario un controllo visivo. Se questo avesse dato esito negativo, sarebbe stata eseguita la procedura di aggiramento del pannello, supponendo di essere sul lato posteriore. Se anche il controllo del secondo lato non avesse avuto successo, si sarebbe ipotizzato di non aver trovato il pannello corretto, bensì un ostacolo simile.

Per svolgere questo controllo si è pensato di utilizzare il classificatore descritto nel capitolo precedente, viste le sue ottime performance anche alla distanza prevista per l'ispezione (compresa in un range tra 50 cm e 1 m dal pannello). Si è quindi creato un nodo ROS che, iscrivendosi ai topic di una delle due telecamere, eseguisse la detection con il classificatore delle chiavi. Analizzando i risultati, si è imposto che se le detection fossero state almeno 5 (lasciando un margine di errore di una unità in meno) e in posizione verosimile, allora l'identificazione del pannello restituiva esito positivo. Come vincoli di posizione si è imposto che avessero dimensioni simili e che le coordinate X e Y fossero distribuite nella stessa zona del pannello. Una volta ottenuto un risultato positivo, gli stessi dati delle detection trovate sono servite per creare una **ROI** (*Region Of Interest*) delle chiavi. Questa zona, al cui interno sono presenti le chiavi trovate, è stata delimitata da un rettangolo di dimensioni opportune (Figura 4.18).

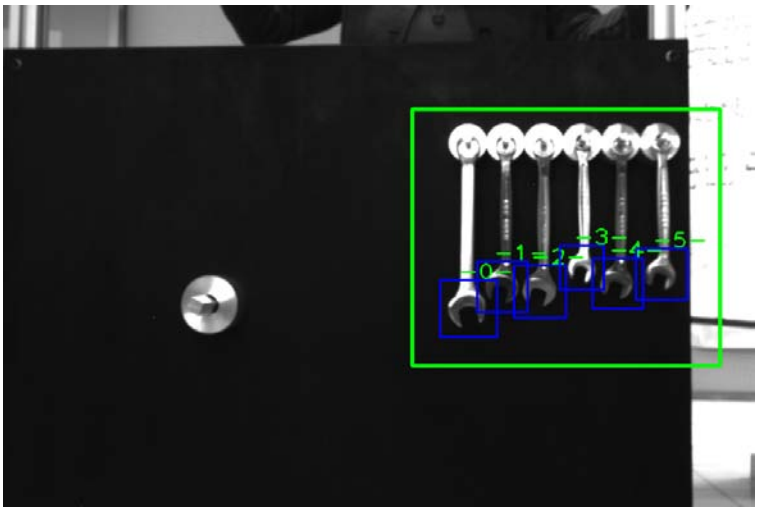


FIGURA 4.18: Ispezione del pannello con localizzazione delle chiavi e creazione della ROI. In verde l'ordinamento per coordinata X crescente.

Passando al 3D, si è scelto di considerare il centroide della parte di point cloud racchiusa nella ROI. Tramite il metodo *pcl::compute3DCentroid*[7] si è quindi ottenuto il centroide della ROI, che ha costituito l'origine di una nuova tf. In questo modo si è potuto muovere il braccio e posizionare il gripper con un certo offset di distanza davanti alle chiavi, pronto per le successive analisi.

4.4.3 Tf sulla chiave

Grazie al nodo ROS per l'ispezione del pannello si è potuto procedere con la fase seguente certi di essere posizionati correttamente, con tutte e sei le chiavi visibili in entrambe le telecamere. In un nuovo nodo ROS si è quindi applicato il classificatore delle chiavi e si è scelto la chiave desiderata in base alle lunghezze relative, come spiegato nella sezione 3.2.4. Si è quindi creato un bounding box approssimativo della chiave, estendendo in verticale il quadrato sulla testa ottenuta dalla detection (Figura 4.19). Si è scelto di non coprire l'estremità superiore della chiave, in quanto la presenza del piolo portava rumore nella point cloud. Filtrando l'intera point cloud originaria, si sono mantenuti solo i punti le cui coordinate X e Y rientravano nell'area selezionata. Grazie a questa operazione i numeri della nuova point cloud da analizzare sono stati ridotti notevolmente, permettendo operazioni più veloci e in tempo reale.

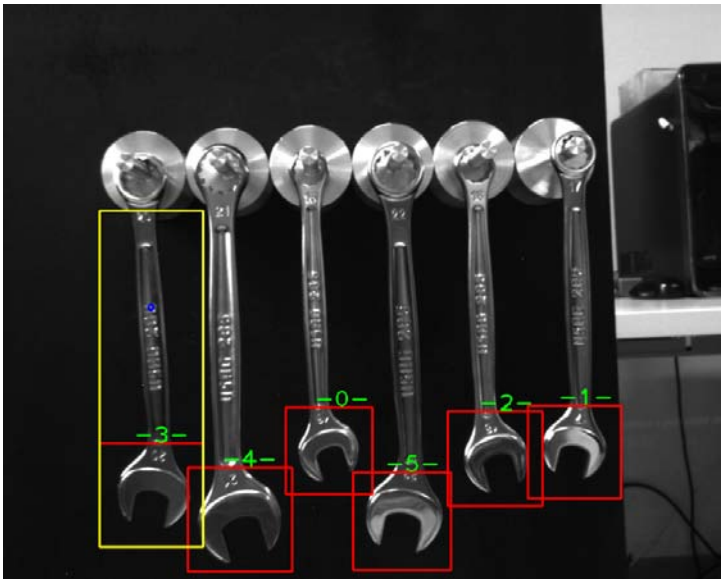


FIGURA 4.19: Bounding box (in giallo) attorno alla chiave e punto centrale per il grasping (in blu).

[7]Module common: http://docs.pointclouds.org/1.0.0/group__common.html

Un primo controllo è stato inserito in questo punto: se i punti della point cloud non risultavano in numero sufficiente, a causa di errori nella detection o nel tracciamento del bounding box, l'intero processo veniva ripetuto. Eliminare un frame e passare a quello successivo è un approccio utile e non problematico, in quanto la point cloud può variare molto da un'acquisizione all'altra. In questo modo si cercava di ridurre il rischio di compiere il movimento di presa del robot in condizioni di potenziale errore.

Se il controllo dava esito positivo, si procedeva ad analizzare la nuova point cloud, contenente parte della chiave e del pannello in background. A seconda delle caratteristiche dello sfondo si sono sviluppati due metodi: infatti il primo pannello da noi costruito presentava una texture non omogenea, che gli permetteva di essere presente e ben visibile nella point cloud. D'altra parte il nuovo pannello, di colore nero uniforme, non era ricostruito nella point cloud, in quanto l'algoritmo di matching non trovava corrispondenze nei punti, eccessivamente indifferenziati. Nel primo caso il programma cercava un **piano tridimensionale** nella point cloud, che rappresentava il pannello di sfondo, con il metodo *pcl::SACSegmentation* descritto nel paragrafo 4.4.1. Individuato lo sfondo, questo veniva eliminato dalla point cloud. I punti rimanenti venivano quindi sottoposti nuovamente alla medesima operazione, e il piano risultante era quello su cui giaceva la chiave. Nel secondo caso, ovvero se lo sfondo non risultava presente nella point cloud, il metodo veniva applicato una sola volta, restituendo subito il piano della chiave.

L'output di questa operazione era rappresentato non solo dai punti della point cloud costituenti la chiave (Figura 4.20), ma anche dalle ben più stabili coordinate geometriche del piano tridimensionale passante per la stessa (Figura 4.21). In questo modo si riusciva a stimare non solo la posizione della chiave nello spazio, ma anche la sua inclinazione, così da facilitare l'operazione di grasping.

Ottenuti questi dati, si è passati a stabilire un punto sullo stelo della chiave su cui posizionare la `tf` per la presa. L'origine della `tf` è stata scelta inizialmente nel 2D, utilizzando come riferimento il bounding box, con le coordinate X e Y centrali rispetto allo stelo (evidenziato in blu nella Figura 4.19). Successivamente, tramite il metodo *projectPixelTo3dRay* descritto nel paragrafo **??**, si è intersecato il raggio tridimensionale passante per il punto 2D con il piano geometrico su cui giace la chiave. Questo ha permesso di ottenere la coordinata Z, e la sua definitiva posizione spaziale.

FIGURA 4.20: Point cloud della chiave nei colori reali, in rosso la point cloud originaria.

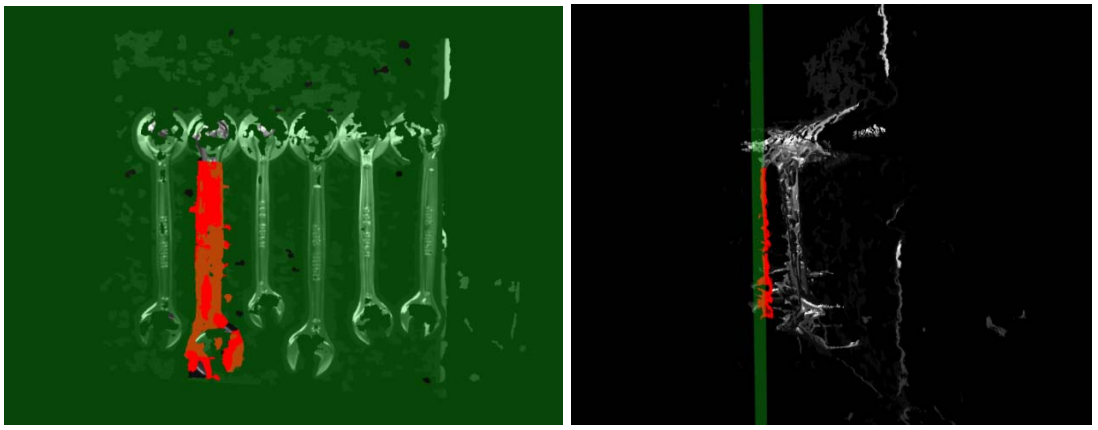


FIGURA 4.21: Point cloud della chiave in rosso e piano passante per essa in verde.

L'ultimo passo necessario per la costruzione della tf richiede la costruzione dei suoi assi. Innanzitutto si è scelto un sistema di riferimento per la chiave concorde con quello dell'end-effector, in modo che l'allineamento delle tf non causasse eccessive rotazioni del braccio. Maggiore precisione è stata aggiunta decidendo di orientare gli assi della tf sulla chiave coerentemente con il piano geometrico della stessa: come risultato due assi giacevano sul piano, mentre il terzo risultava perpendicolare ad esso.

La tf creata (Figura 4.22) è pronta per essere assegnata come goal al sistema, che può così iniziare la fase di grasping .

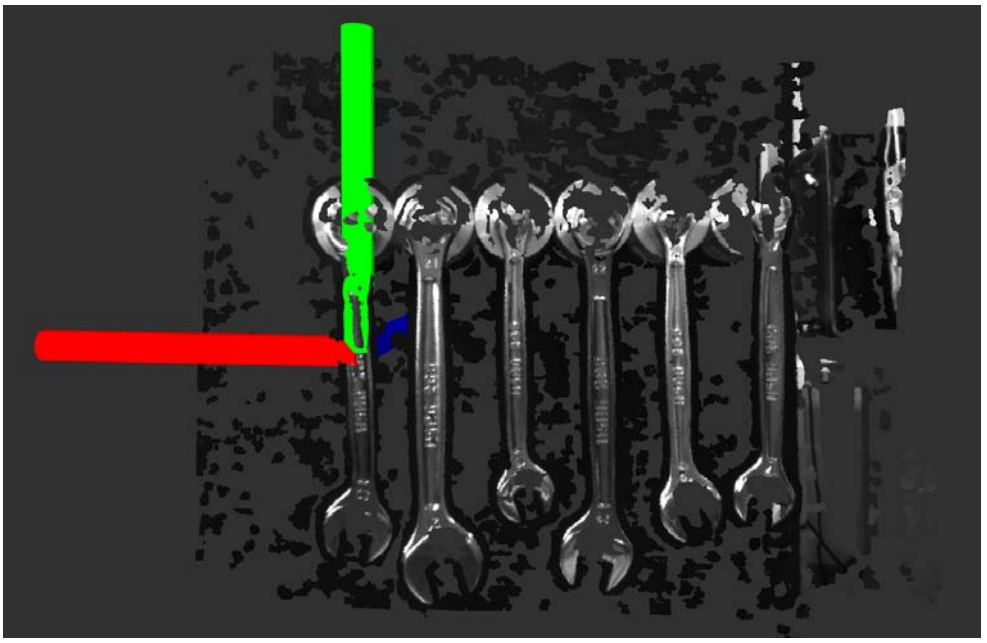


FIGURA 4.22: Tf posizionata al centro dello stelo della chiave.

Localizzazione dei pioli

Per perfezionare maggiormente il posizionamento del punto di presa, si è sviluppato un programma in grado di rilevare i pioli su cui erano appese le chiavi. Localizzati i pioli, o meglio la loro base circolare, si poteva assumere con una buona approssimazione che al loro centro fosse posizionato il bordo superiore della chiave. In questo modo si riusciva ad ottenere una più precisa coordinata superiore del bounding box racchiudente l'intera chiave, descritto nel paragrafo precedente. Di conseguenza il nuovo centro, su cui è costruita la `tf` per il grasping, risultava posizionato all'altezza giusta per ogni chiave.

I problemi riscontrati nel localizzare le basi dei pioli sono stati dovuti principalmente alla presenza di riflessi, essendo di metallo riflettente, e di rumore causato dai pioli stessi e dalle chiavi appese. L'approccio scelto è stato di utilizzare la trasformata di Hough per le figure circolari, più precisamente l'implementazione di OpenCV **HoughCircles**.

```
void HoughCircles (InputArray image, OutputArray circles, int method, double
dp, double minDist, double param1=100, double param2=100, int minRadius=0,
int maxRadius=0 )
```

- **-image**: immagine di input a 8-bit, single-channel e grayscale;

- **-circles**: vettore di output di elementi costituiti da 3 float (x, y, radius), che definiscono ogni cerchio trovato.
- **-method**: metodo di detection, l'unico implementato attualmente è CV_HOUGH_GRADIENT, descritto in [34];
- **-dp**: rapporto tra risoluzione dell'immagine e risoluzione dell'accumulatore. Ad esempio, se dp=2 l'accumulatore ha dimensioni dimezzate rispetto all'immagine;
- **-minDist**: distanza minima tra i centri dei cerchi rilevati;
- **-param1**: valore di soglia superiore utilizzato nel Canny edge detector (il valore di soglia inferiore è due volte più piccolo);
- **-param2**: valore di soglia dell'accumulatore per i centri dei cerchi nella fase di detection. Minore è il valore, più cerchi "falsi" possono essere trovati;
- **-minRadius**: raggio minimo dei cerchi;
- **-maxRadius**: raggio massimo dei cerchi.

HoughCircles opera pressapoco allo stesso modo di *HoughLines*, ma invece di restituire i due parametri $(r,\ \theta)$ che definiscono una linea, restituisce in un vettore di float i tre parametri necessari per definire un cerchio:

$$C : (x_{center},\ y_{center},\ r),$$

dove $(x_{center},\ y_{center})$ sono le coordinate del centro e r il raggio.

Per identificare i sei pioli si è applicato questo metodo settando i vari parametri in modo da preferire la presenza di qualche falso positivo, eliminabile nella fase di post-processing, piuttosto che non rilevare qualche piolo. Come soglia superiore del Canny edge-detector si è scelto il valore di 150, in grado di evidenziare tutti i contorni dei pioli in varie situazioni luminose. Si è imposto inoltre un limite inferiore e superiore ai raggi dei cerchi, in proporzione alle dimensioni dell'immagine, eliminando detection non verosimili. A questo punto si sono analizzate le varie detection restituite: la geometria del sistema (Figura 4.23) ci ha permesso di imporre numerosi vincoli, cercando sei detection affiancate, alla stessa altezza, e di raggi uguali.

Per ottenere ciò si sono suddivise più volte le detection in **cluster**:

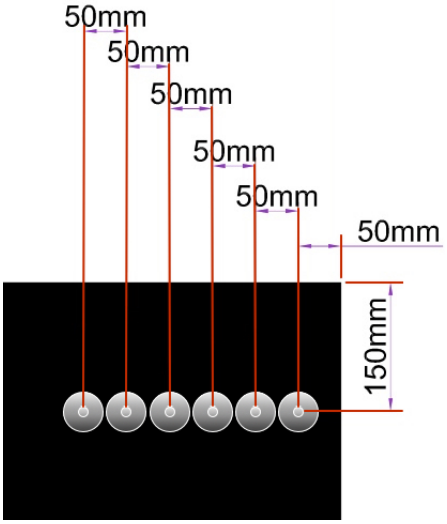


FIGURA 4.23: Posizione dei pioli nel pannello.

1. *Clustering in base alla posizione del centro*: a causa dei riflessi sulla base metallica dei pioli e della presenza dell'apertura circolare delle chiavi appese, si è notato un gran numero di detection concentriche su ogni piolo (Figura 4.24). Per questo si sono raggruppate le detection con centri vicini, mantenendo poi solo il cerchio di raggio maggiore per ogni cluster.

FIGURA 4.24: Detection concentriche pre-clustering.

2. *Clustering in base al raggio*: si sono raggruppati i cerchi con le medesime dimensioni, mantenendo solo i cluster con più di sei elementi;

3. *Clustering in base alla coordinata Y*: per ogni cluster precedente si sono suddivise le detection a seconda del posizionamento in altezza nell'immagine, mantenendo poi solo i cluster con più di sei elementi;

4. *Selection*: se il passo precedente ha restituito più di un cluster, si è scelto di selezionare quello posizionato più in alto nell'immagine. Infatti si è osservato sperimentalmente che eventuali false detection sono rilevate nella parte di immagine inferiore, dove sono presenti i contorni rumorosi delle chiavi e di eventuali ombre; sopra i pioli invece l'omogeneità del pannello nero non crea contorni e di conseguenza false detection.

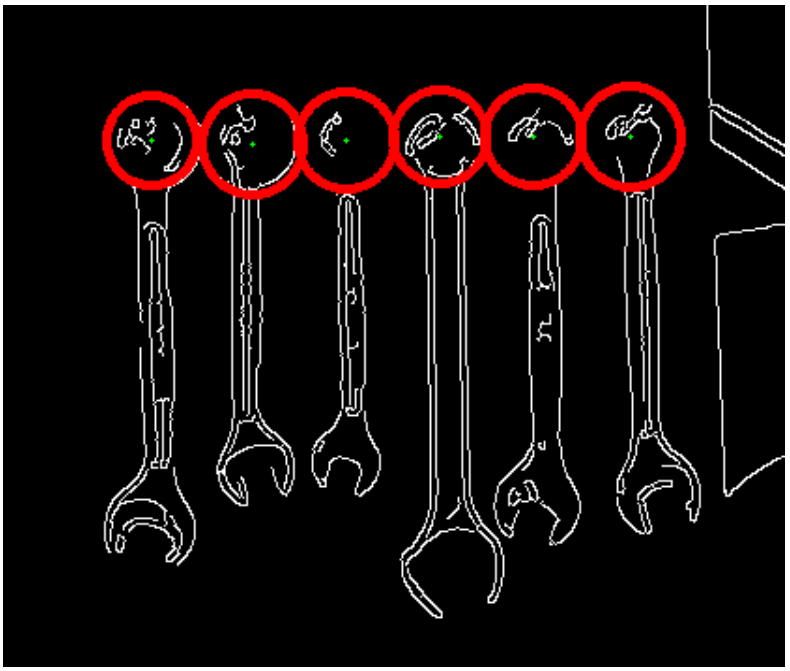


FIGURA 4.25: Identificazione corretta dei pioli.

Se un qualsiasi passaggio precedente non trovava abbastanza risultati, l'intero processo di detection veniva ripetuto anche più volte, abbassando il valore dell'accumulatore di HoughCircles (*param2*) e della soglia superiore di Canny (*param1*). In questo modo si ottengono rispettivamente più detection e più contorni.

Inclinazione della testa della chiave

Definito il punto della chiave su cui viene effettuata la presa, si è passati a localizzare il centro dell'apertura della chiave e la sua inclinazione. Queste due informazioni sono fondamentali per poter eseguire la successiva fase di inserimento sulla valvola.

A questo scopo si è rivelato utile il bounding box restituito dal classificatore delle chiavi, che ha permesso di restringere il campo di analisi alla sola testa delle chiavi. L'approccio scelto è stato di identificare le due linee parallele dell'apertura della chiave tramite l'implementazione probabilistica di OpenCV della Trasformata di Hough per le linee, **HoughLinesP**. In questo modo non solo si ottiene l'inclinazione, ma è anche facilmente individuabile il punto centrale equidistante da esse.

```
void HoughLinesP (InputArray image, OutputArray lines, double rho, double
theta, int threshold, double minLineLength=0, double maxLineGap=0 )
```

- **-image**: immagine di input a 8-bit, single-channel e binaria;
- **-lines**: vettore di output contenente le linee trovate. Ogni linea è rappresentata da un vettore di 4 elementi $(x_1,\ y_1,\ x_2,\ y_2)$, dove $(x_1,\ y_1)$ e $(x_2,\ y_2)$ sono gli estremi del segmento trovato.
- **-rho**: risoluzione della distanza dell'accumulatore in pixels;
- **-theta**: risoluzione angolare dell'accumulatore in radianti;
- **-threshold**: parametro di soglia dell'accumulatore. Solo le linee con $voto > threshold$ sono restituite;
- **-minLineLength**: lunghezza minima delle linee. I segmenti più corti di questo valore sono rifiutati;
- **-maxLineGap**: massimo intervallo permesso tra punti per considerarli della stessa linea.

Il problema principale ha riguardato la presenza dei riflessi di luce sulla chiave (Figura 4.26) e le ombre sul pannello di sfondo (Figura 4.27). Il Canny edge detector applicato nell'implementazione di HoughLinesP rilevava infatti molti contorni errati, non solo

quelli reali del bordo della chiave. Un perfetto settaggio dei parametri volto ad eliminare questo rumore non era possibile, in quanto le variazioni di luce ed esposizione impedivano di identificare dei valori fissi da assegnare ai parametri.

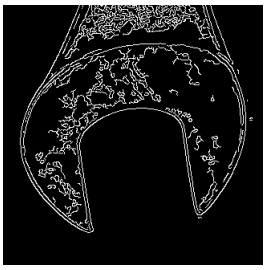


FIGURA 4.26: Rumore dovuto ai riflessi di luce sulla chiave.

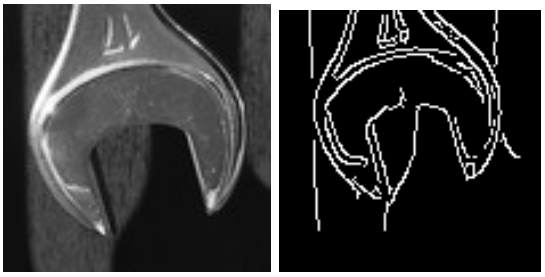


FIGURA 4.27: Rumore dovuto ai riflessi di luce e alle ombre sul pannello.

Tra questo rumore si è quindi cercato di identificare i reali bordi dell'apertura tramite vincoli applicati nel post processing. Il primo riguardava l'inclinazione delle linee: infatti le caratteristiche geometriche delle chiavi combinate impongono un'angolazione dell'apertura rispetto allo stelo di 15°, a destra o a sinistra. Assumendo di essere posizionati con le telecamere di fronte alle chiavi, e paralleli al pannello, si è scelto di cercare solo le linee con questa angolazione, a meno di $\pm 20°$ di errore. In seguito all'applicazione del metodo HoughLinesP si sono quindi scartate tutte le linee con angolazione differente.

A questo punto si sono cercate coppie di **linee parallele**, come da geometria dell'apertura della chiave. Anche qui si è utilizzato un margine d'errore sull'inclinazione dei segmenti, più precisamente di $\pm 7°$. Per ogni coppia di linee parallele trovata si è poi calcolata la distanza tra di esse. Poiché l'immagine su cui viene fatta la detection è la parte delimitata dal bounding box restituito dal classificatore, si è potuto imporre un range di dimensioni probabili entro cui rientra l'apertura della chiave. Quindi, invece di fissare dei limiti in pixel, il range è stato creato in proporzione alle dimensioni del bounding box. Il vincolo imposto è stato che la distanza fra i due segmenti paralleli fosse compresa tra *image.cols/5* e *image.cols/6*5*. Se dopo questa operazione il risultato non

fosse stato univoco, di una sola coppia, si è pensato di selezionare la coppia vincente tramite un ulteriore vincolo. Analizzando le immagini infatti si è notato come il rumore e quindi eventuali linee errate fossero disposte in particolare sull'area metallica della testa della valvola; nella zona centrale dell'apertura invece, essendo visibile il pannello con pattern uniforme, non era rilevato nessun contorno e quindi nessuna linea. Si è quindi scelta la coppia di segmenti paralleli più vicina al centro (Figura 4.28), assumendo che le altre coppie, essendo più lontane, fossero false detection dovute ai riflessi di luce sulla testa.

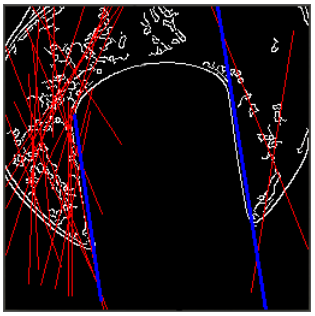


FIGURA 4.28: Individuazione delle linee parallele della chiave (in blu) tra quelle restituite da HoughLinesP (in rosso).

Ottenuto un risultato univoco rappresentato da due segmenti paralleli, si è calcolata matematicamente l'**inclinazione dell'apertura** eseguendo una media fra i coefficienti angolari delle due linee. Sempre matematicamente si è trovato un **punto centrale** equidistante da esse, rappresentante il centro dell'apertura della testa della chiave (Figura 4.29).

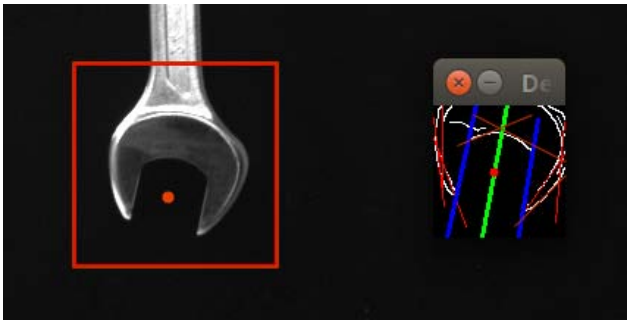


FIGURA 4.29: Individuazione del centro (in rosso) equidistante dalle linee parallele trovate (in blu).

Da questo punto bidimensionale si è passati al punto nello spazio 3D tramite il metodo *projectPixelTo3dRay*, descritto nel paragrafo 4.4.1. Si è quindi creata una `tf`, con origine in questo punto e assi orientati secondo l'inclinazione dell'apertura appena trovata.

4.4.4 Tf sulla valvola

Localizzazione della valvola

Effettuato il grasping della chiave, il task successivo consisteva nella localizzazione della valvola e nell'inserimento della chiave nella stessa. Successivamente alla presa tuttavia il braccio si trovava eccessivamente vicino al pannello, e il limitato angolo di visione delle telecamere impediva di visualizzare la valvola. Si è quindi inserito un movimento hard-coded per allontanare il braccio dal pannello e spostarlo nella direzione della valvola, così da inquadrarla con certezza.

Un problema che si è potuto osservare in questa fase è che l'ampia superficie metallica della chiave e dello stelo creavano un numero di riflessi tale da impedire la costruzione di una point cloud stabile e veritiera. Inoltre le specifiche descriventi la valvola che sarebbe stata trovata in gara indicavano solo forma e dimensioni, e un generico colore argentato. Per questi motivi si è scelto di non utilizzare il sistema tridimensionale della point cloud per ogni task riguardante la valvola, bensì il più semplice metodo di **triangolazione** su singoli punti, identificati nelle due immagini.

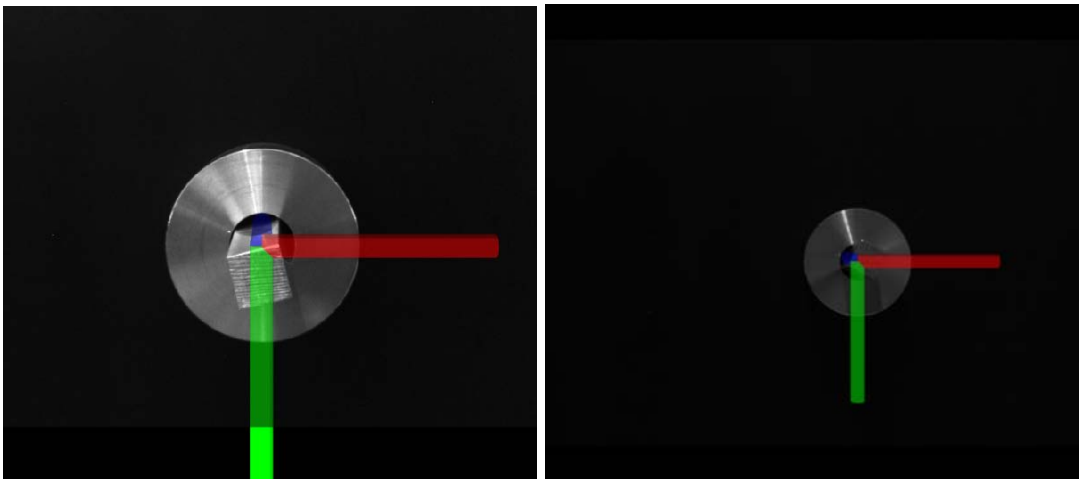
Si è quindi proceduto lanciando il classificatore della valvola sia sull'immagine sinistra sia su quella destra (Figura 4.30). Poiché la triangolazione deve essere fatta sulla stessa feature, identificata univocamente su entrambe le immagini, si è scelto di prendere il centro del bounding box restituito dal classificatore. Infatti le dimensioni del quadrato, a causa di come è strutturato il metodo di detection di OpenCV, cambiano leggermente di frame in frame, mantenendo tuttavia il medesimo centro. Sfruttando i parametri noti delle telecamere si è potuto triangolare sul centro della valvola, se trovato in entrambe le immagini.

Ottenuto il punto nello spazio, si è creata una `tf` sul centro della valvola (Figura 4.31), con assi orientati coerentemente con l'end-effector (che si è supposto essere di fronte al pannello). Questa `tf` serve per posizionare il braccio (e le telecamere) nella posizione di pre-inserimento, precisamente allineato davanti alla valvola e alla distanza desiderata.

(a)

(b)

FIGURA 4.30: Localizzazione della valvola su immagine sinistra e destra, e triangolazione sul centro.



(a) (b)

FIGURA 4.31: Tf posizionata al centro della valvola, con assi coerenti all'end-effector.

Inclinazione dello stelo della valvola e tf per l'inserimento

Una volta posizionati di fronte alla valvola lo stelo della stessa è quasi perfettamente un quadrato nell'immagine bidimensionale, non essendoci forte distorsione prospettica laterale. Per individuare l'inclinazione dello stelo si è scelto di utilizzare il metodo *HoughLinesP* come per la testa della chiave. L'obbiettivo è trovare le linee che delimitano il quadrato dello stelo, per ricavarne l'orientazione.

Come per la chiave, si è sfruttato il bounding box restituito dal classificatore per ispezionare solo la superficie di immagine occupata dalla valvola. La maggiore area riflettente rispetto alla chiave porta ancora più rumore nei contorni, a causa del gran numero di riflessi. Tuttavia, si possono imporre numerosi vincoli per identificare correttamente il quadrato.

L'approccio ideato è costituito dalle seguenti fasi:

1. **Bilateral Filtering:** i classici filtri (*Gaussian, Median* etc.) che vengono usati per ridurre il rumore hanno lo svantaggio di sfumare anche gli *edge* (bordi). Nel task in questione questo effetto è fortemente penalizzante, così si è scelto di utilizzare il *bilateral filter* [24]. Analogamente al Gaussian filter ai pixel vicini vengono assegnati dei pesi, ma viene tenuta in considerazione anche la similarità. Come si può vedere nella Figura 4.32, rappresentante un edge che divide una zona chiara da una scura, le due parti di immagine vengono sfumate separatamente, preservando il bordo.

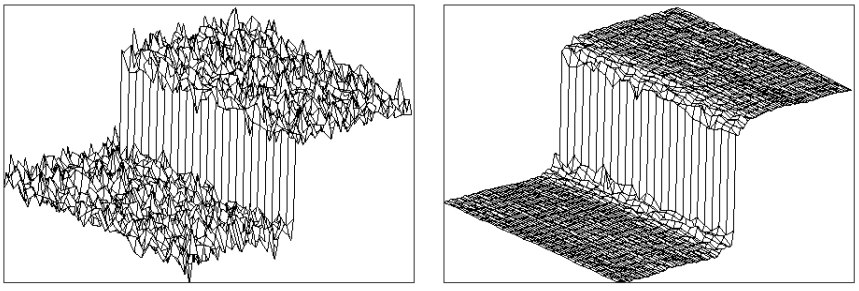


FIGURA 4.32: Applicazione del Bilateral Filter su un *edge*.

Applicato il Bilateral Filter sulla valvola (Figura 4.33.a), si può notare che gli edge rimangono ben definiti, mentre il rumore è stato ridotto notevolmente (Figura 4.33.b);

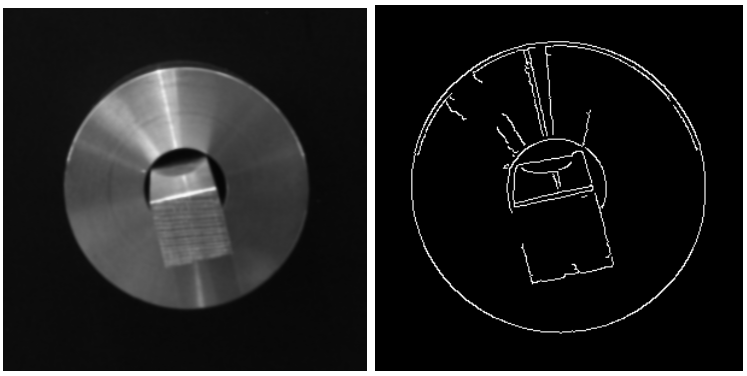


FIGURA 4.33: Applicazione del Bilateral Filter sulla valvola.

2. **HoughLinesP:** filtrata l'immagine per ridurre il rumore, si è applicato questo metodo, descritto in precedenza, per identificare le linee;

3. **Prolungamento:** per garantire l'incidenza dei segmenti del quadrato, descritta nel prossimo punto, ogni segmento restituito dal metodo precedente è stato prolungato in entrambi i versi di 1/3 della sua lunghezza;

4. **Incidenza e perpendicolarità (1° ciclo):** per ogni coppia di linee possibile si calcola geometricamente se sono incidenti. In caso positivo, si controlla l'angolo di incidenza e si mantengono solo le coppie perpendicolari;

5. **Incidenza e perpendicolarità (2° ciclo):** per ogni coppia trovata nel passo precedente si cerca un altro segmento incidente e perpendicolare ad una delle due linee (Figura 4.34);

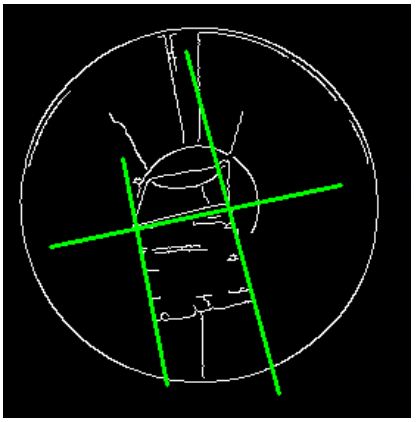


FIGURA 4.34: Identificazione dei tre lati del quadrato.

6. **Distanza:** i tre segmenti vengono salvati in un nuovo vettore solo se la distanza tra i due punti di incidenza risulta entro un range pre-determinato. Questo è stato

stabilito in proporzione al bounding-box, e quindi alle dimensioni della valvola, e rappresenta le dimensioni che si prevede abbia il lato del quadrato dello stelo;

7. **Incidenza e perpendicolarità (3° ciclo):** per ogni tripletta di segmenti restituita dal passaggio precedente si cerca una quarta linea, incidente e perpendicolare alla prima e ultima. In questo modo le quattro linee vanno a formare un quadrato per costruzione (Figura 4.35);

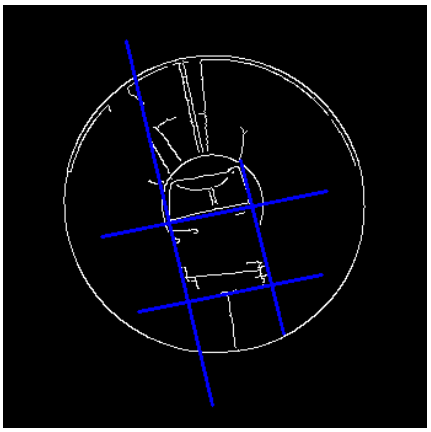


FIGURA 4.35: Identificazione dei quattro lati del quadrato.

8. **Distanza:** un ultimo controllo viene fatto ancora sulla distanza tra i nuovi punti di incidenza, che deve essere simile a quella trovata in precedenza. Questo check assicura che i lati abbiano la stessa lunghezza, costituendo un quadrato;

In seguito a numerose prove sperimentali, si è osservato che l'individuazione del quarto lato risultava difficoltosa se il contrasto luminoso non era sufficientemente alto. Si è così deciso di rendere opzionale l'ultimo passaggio: se il quarto lato non veniva trovato, veniva restituita la prima tripletta trovata al passaggio precedente.

Individuato il contorno del quadrato dello stelo, si sono calcolate le due inclinazioni dei lati. Per ridurre le rotazioni da effettuare con il gripper per l'inserimento, si è scelta l'inclinazione più vicina alla verticale (Figura 4.36).

A questo punto si è creata una nuova `tf` (Figura 4.37) a partire da quella precedente, indicante il centro della valvola. Le operazioni compiute sono state le seguenti:

- Uno spostamento lungo l'asse Z, indicante la profondità, di -6 cm. La `tf` in questo modo è stata allontanata dal pannello e posizionata sullo stelo, sempre al centro della valvola;

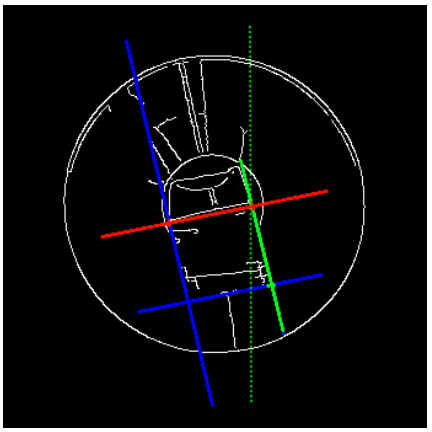
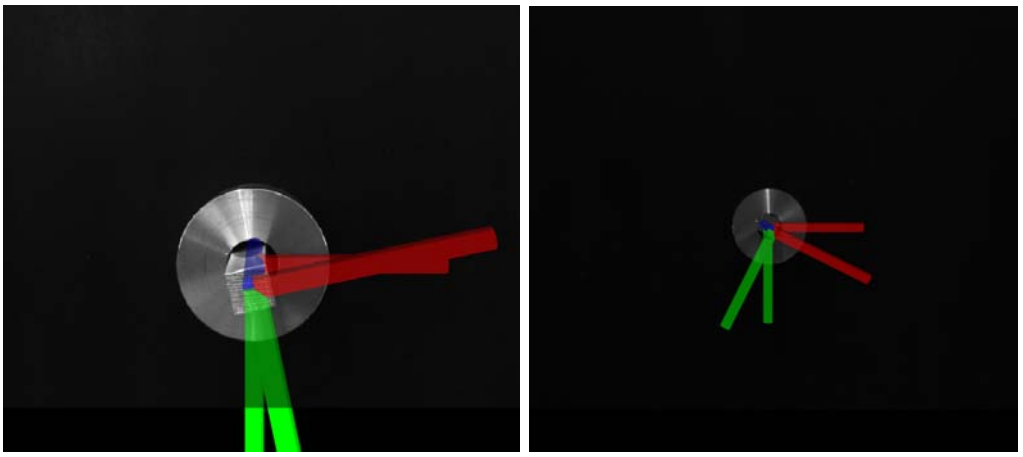


FIGURA 4.36: Selezione dell'inclinazione più vicina alla verticale (verde).

- Una rotazione degli assi X e Y attorno a Z, ora coerenti con l'inclinazione dello stelo trovata in precedenza.



(a) (b)

FIGURA 4.37: In primo piano la tf per l'inserzione con gli assi inclinati coerentemente allo stelo.

La nuova tf viene quindi passata come *goal* al sistema, che provvederà a sovrapporci la tf sull'apertura della chiave, muovendo il braccio di conseguenza.

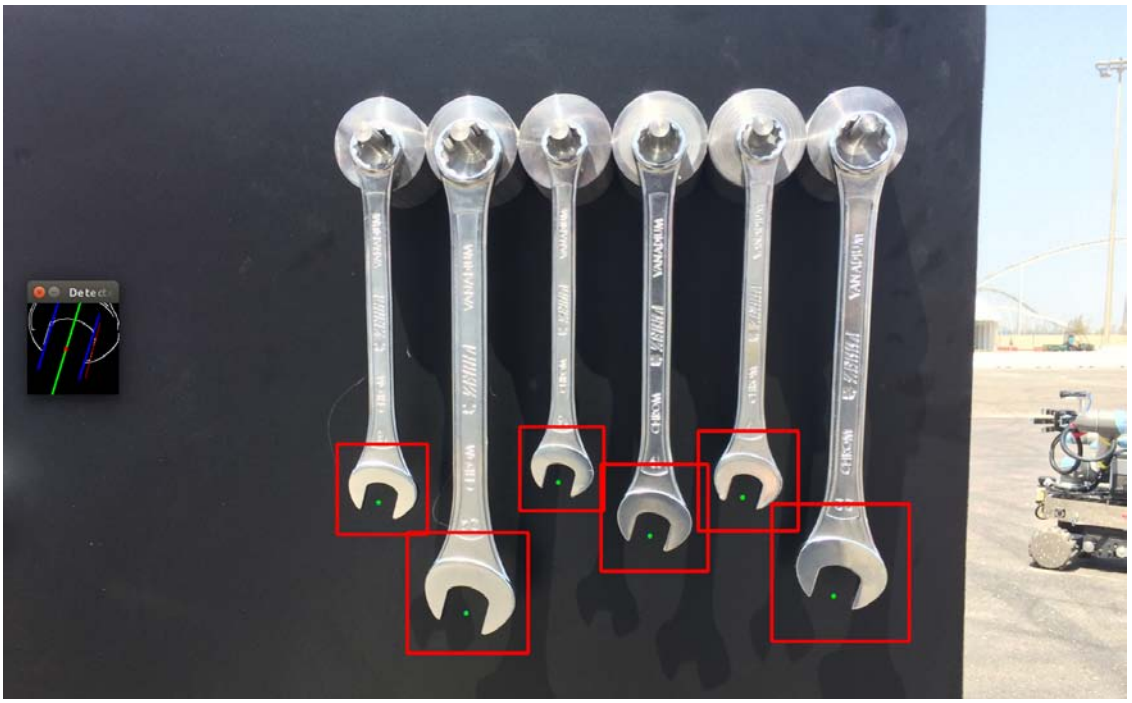
4.5 Risultati

Tf sullo stelo della chiave

La corretta individuazione del punto centrale dello stelo della chiave, su cui effettuare il grasping, è dipendente dalla bontà della point cloud generata. Se questa risulta instabile o parziale, come può succedere a causa di un errato tuning dei parametri, di una distanza differente dal pannello rispetto a quella prevista, o per riflessi di luce variabili ed intensi, la `tf` non può essere posizionata correttamente. Tuttavia, nelle prove effettuate con una point cloud sufficientemente buona, il metodo sviluppato riesce ad ottenere ottimi risultati, permettendo il grasping della chiave selezionata.

Tf sull'apertura della chiave

Per quanto riguarda la corretta individuazione del centro dell'apertura della chiave, di fondamentale importanza per l'inserimento sulla valvola, si è testato l'algoritmo sulle immagini di chiavi raccolte in gara. Su **225 chiavi** presenti, il centro è stato correttamente localizzato in **209 chiavi**, con una precisione del **92,9%** (Figura 4.38).



(a)

(b)

FIGURA 4.38: Individuazioni corretta del centro dell'apertura delle chiavi in gara.

Da notare che il programma ha funzionato anche in condizioni ben diverse da quanto previsto, come mostrano la Figura 4.39 a distanza ravvicinata e la Figura 4.40 con una forte inclinazione laterale.

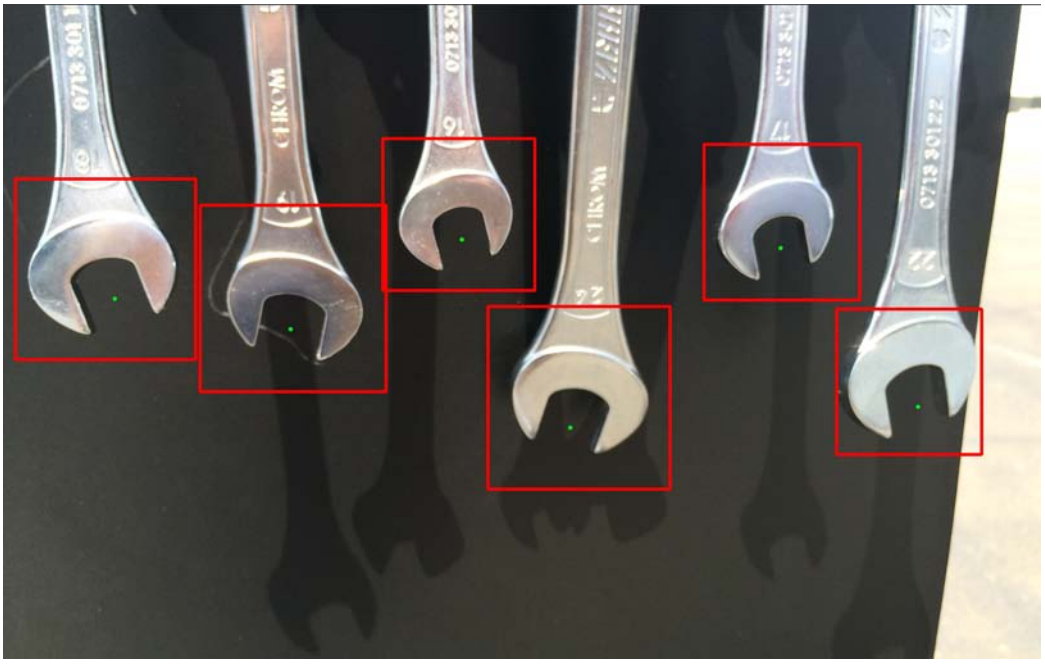


FIGURA 4.39: Localizzazione del centro dell'apertura delle chiavi in gara a distanza ravvicinata.

FIGURA 4.40: Localizzazione del centro dell'apertura delle chiavi in gara con forte inclinazione

Tf sulla valvola

Grazie alle ottime performance del classificatore, la `tf` creata sulla valvola viene creata correttamente. Un problema tuttavia si può presentare se la valvola non dovesse essere interamente visibile in entrambe le immagini, destra e sinistra. In questo caso la triangolazione non dà esito positivo, e l'unica soluzione è muovere il robot in una posizione migliore.

Per quanto riguarda la tf orientata, essa è dipendente dalla corretta individuazione del quadrato dello stelo. Questa a sua volta richiede un settaggio dell'esposizione e del contrasto delle telecamere tale da garantire immagini bilanciate, con i lati dello stelo sufficientemente contrastati.

Capitolo 5

Object Tracking e Visual Servoing

In questo capitolo è descritto il lavoro svolto e i test che sono stati fatti per implementare tecniche di object tracking e visual servoing nella challenge MBZIRC. Nonostante essi non siano stati utilizzati effettivamente in gara, presentano spunti interessanti per una possibile applicazione futura.

ViSP è la libreria che, insieme ad OpenCV, è stata scelta dalla nostra squadra per i task di visione del progetto. Fornisce numerosi algoritmi progettati specificatamente per l'object tracking e il visual servoing.

L'**object tracking** è un insieme di tecniche che permette di *seguire* un determinato oggetto in un insieme di immagini o in un video. Possono verificarsi due casi: che sia l'oggetto a muoversi nello spazio, o il sensore visivo stesso. Ad ogni modo, prima di iniziare il tracking l'oggetto deve essere identificato e localizzato nella prima immagine (o frame): in seguito, per ogni immagine successiva verrà cercata la nuova posizione dell'oggetto.

Il **visual servoing** consiste nel controllare il movimento di un sistema robotico usando informazioni visive come feedback. Richiede di lavorare in sinergia con l'object tracking, da cui ottiene le informazioni di posizione dell'oggetto da raggiungere o seguire. A causa della varietà dell'hardware (robot e sensori), è stato estremamente difficile creare un ambiente software che permettesse un modello veloce e portabile di visual servoing. ViSP è stata la prima libreria ad offrire queste caratteristiche: semplicità, indipendenza rispetto all'hardware e portabilità[19].

5.1 Object Tracking in ViSP

La libreria ViSP fornisce cinque algoritmi di *object tracking*, con diverse caratteristiche per adattarsi ad ogni circostanza:

1. **Blob tracker**[28]: viene usato quando l'oggetto è una regione dell'immagine con lo stesso livello di grigio, di preferenza nero su uno sfondo bianco o bianco su uno sfondo nero (Figura 5.1).

FIGURA 5.1: Blob tracker.

2. **Keypoint tracker**[27]: sfrutta il tracker *KLT* di OpenCV, una implementazione del *Kanade-Lucas-Tomasi feature tracker*[17][25]. Riconosce caratteristiche utili, chiamate *keypoint*, grazie all' *Harris corners detector*, e le rintraccia nelle immagini successive (Figura 5.2).

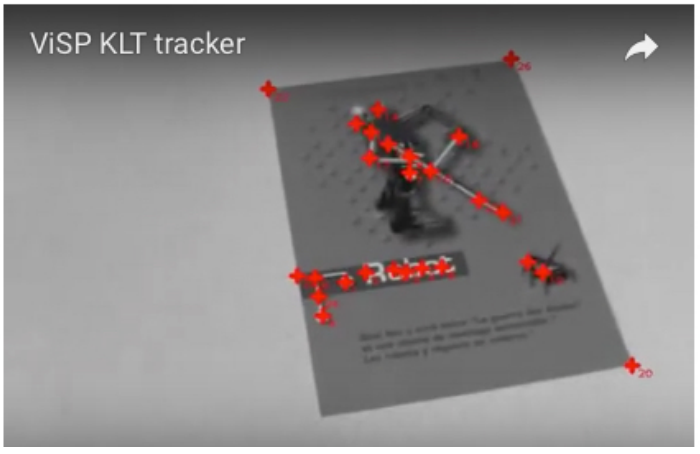


FIGURA 5.2: KLT tracker.

3. **Moving-edge tracker**[31]: segue linee o ellissi usando i moving-edges. Dalla precedente posizione di un moving edge, il tracker lo cerca lungo la normale del contorno entro un certo range (Figura 5.3).

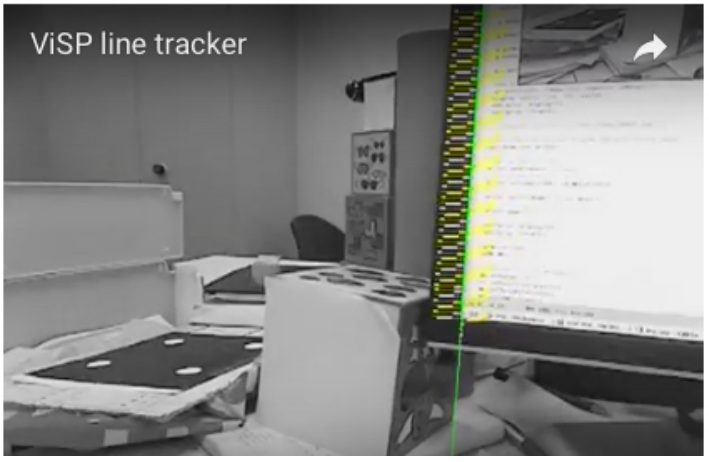


FIGURA 5.3: Moving-edge tracker.

4. **Model-based tracker**[30]: permette di eseguire il tracking di un oggetto sfruttando la conoscenza del suo modello CAD e fornendo la sua localizzazione 3D (la posa dell'oggetto espressa nel frame della telecamera). ViSP implementa tre diversi model-based tracker a seconda delle visual feature utilizzate:

   - `vpMbEdgeTracker` si basa sulle feature moving-edges;
   - `vpMbKitTracker` considera i KLT keypoint su ogni lato visibile del modello;
   - `vpMbEdgeKitTracker` è una versione ibrida che usa sia i moving-edges sia i KLT keypoint (Figura 5.4).

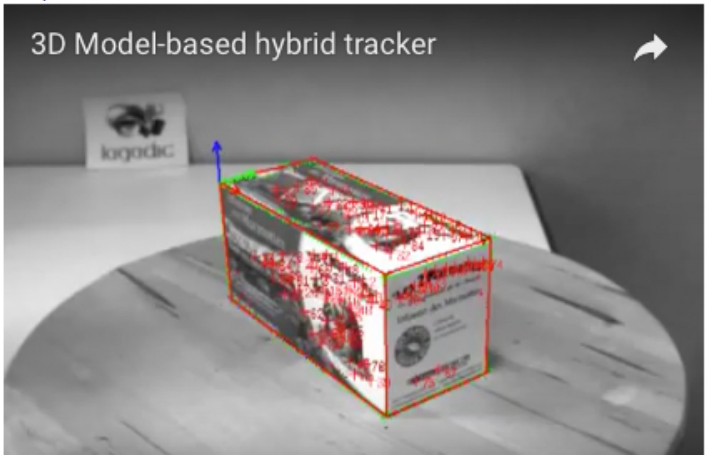


FIGURA 5.4: Model-based tracker ibrido.

5. **Template tracker**[32]: diversamente dagli altri tracker sfrutta algoritmi di *image registration*[6] invece di basarsi sulle visual feature. Il tracker stima la trasformazione tra il template di riferimento e la sua posizione corrente nelle immagini (Figura 5.5).

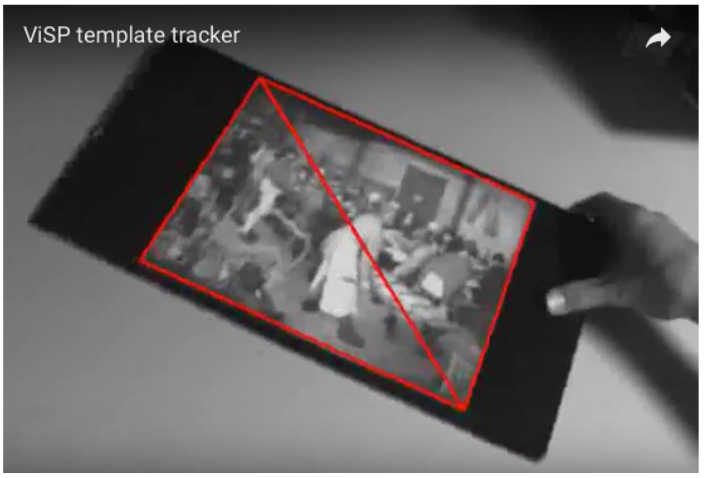


FIGURA 5.5: Template tracker.

In ViSP sono state implementate tre diverse *similarity function*[33]:

- **Sum of Square Differences** (`vpTemplateTrackerSSD` class)
- **Zero-mean Normalized Cross Correlation** (`vpTemplateTrackerZNCC` class)
- **Mutual Information** (`vpTemplateTrackerMI` class)

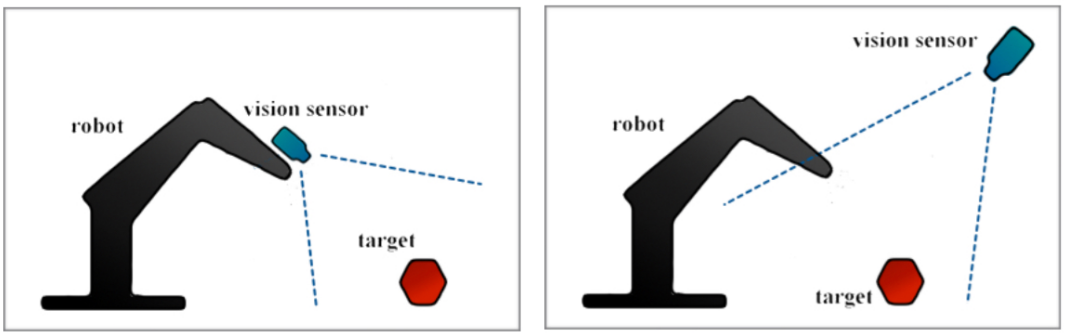
Ognuna di queste funzioni può essere utilizzata in quattro modi diversi:

- *Inverse Compositional*;
- *Forward Compositional*;
- *Forward Additional*;
- *Efficient Second-order Minimization (ESM).*

5.2 Visual Servoing

In una applicazione di visual servoing le immagini possono essere acquisite da una o più telecamere posizionate sull'end-effector del robot o su postazioni fisse. Quando il sensore visivo è posizionato sull'end-effector mobile del robot, si ha una configurazione

eye-in-hand(Figura 5.6 (a)); viceversa, una configurazione **eye-to-hand** presenta una telecamera fissa che osserva sia il robot che la scena (Figura 5.6 (b)).



(a) (b)

FIGURA 5.6: Configurazione *eye-in-hand* (a) e *eye-to-hand* (b).

Scelta la configurazione, è necessario selezionare un insieme di *visual feature* **s**, che in combinazione con la loro posizione desiderata **s*** permette di creare una specifica **legge di controllo** (Figura 5.7). Questa legge di controllo garantisce la convergenza di **s** alla sua destinazione **s***, minimizzando il vettore di **errore** $\boldsymbol{e} = (\boldsymbol{s}^* - \boldsymbol{s})$.

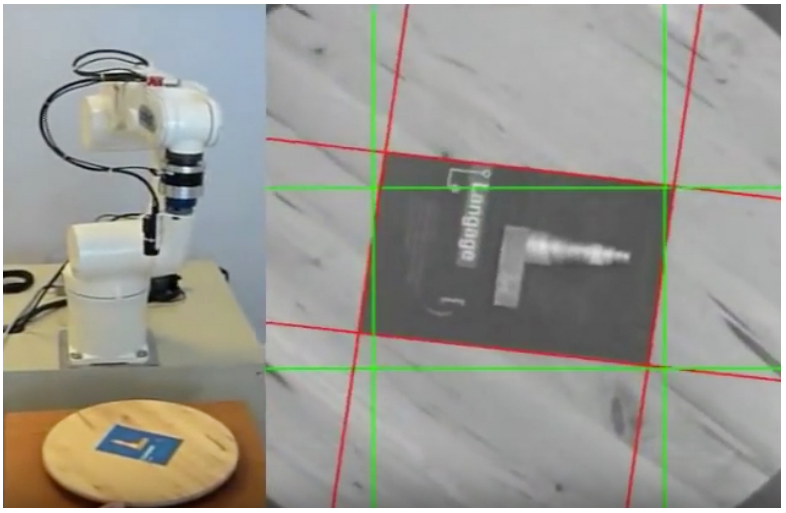


FIGURA 5.7: Esempio di visual servoing: in rosso il set di feature correnti **s**, in verde il set desiderato **s***.

Le tecniche di visual servoing possono essere classificate in due principali tipologie in base alle visual feature adottate:

- **Image-based** (IBVS): le feature sono descritte con coordinate bidimensionali sul piano immagine, pertanto viene anche chiamato *2D servoing*.

- **Position/pose based** (PBVS): le feature sono utilizzate per stimare informazioni 3D come la posa del target, per questo viene chiamato *3D servoing*.

Sono stati sviluppati anche alcuni approcci ibridi, come *2D 1/2 servoing* [3].

5.3 Approccio per MBZIRC

5.3.1 Object Tracking

Analizzando le caratteristiche degli object tracker forniti da ViSP e la loro applicabilità nel caso della challenge, si è deciso di testare il *moving-edges* e il *template-based tracker*.

Il **blob tracker** è stato scartato poiché le chiavi non sono regioni stabili di immagine con lo stesso livello di grigio, quindi non sono facilmente convertibili in blob.

Nemmeno il **keypoint tracker** è stato preso in considerazione poiché, a causa dei numerosi riflessi di luce sulle chiavi, i keypoint che possono essere rilevati in un'immagine possono cambiare drasticamente nelle immagini successive.

Per quanto riguarda il **model-based tracker**, sono emersi subito numerosi problemi: innanzitutto, questo tracker avrebbe richiesto un modello CAD della chiave, che come mostrato nel paragrafo 3.2 non è precisamente disponibile a causa della variabilità dei parametri di forma. Un altro problema riguardava la necessità di fornire un file di inizializzazione in input, con coordinate 3D di alcuni punti usate per computare una posa iniziale, punti le cui coordinate 2D nell'immagine dovevano essere selezionate successivamente. Questa fase di inizializzazione si è dimostrata essere troppo problematica, dato soprattutto il carattere autonomo della challenge che impediva input da utente.

Scartati questi tracker, si è analizzato il **moving-edges tracker**. L'ipotesi iniziale era di utilizzare le due linee parallele dello stelo della chiave come moving-edges. Come input, questo tracker richiede due punti per ogni linea, e il *tuning* di alcuni parametri. Per esempio, è possibile impostare il range lungo la normale del contorno nel quale il tracker cerca la nuova posizione del moving-edge. Per testare questo tracker è stato sviluppato un programma di test in grado di acquisire immagini live da una webcam ed eseguire il tracking successivamente all'input dell'utente (Figura 5.8).

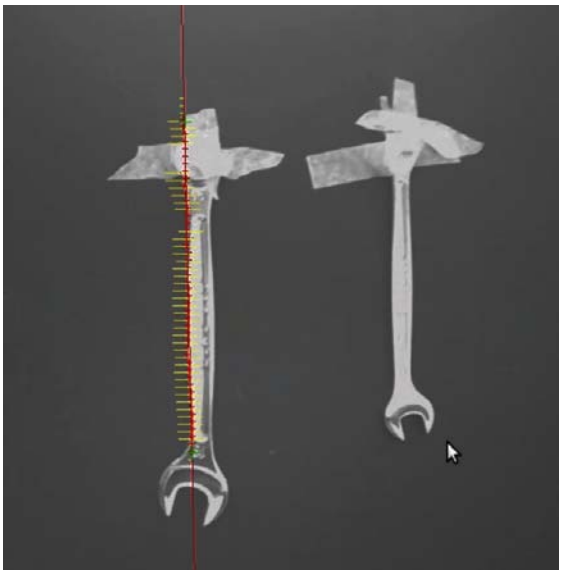


FIGURA 5.8: Moving-edge tracker testato su una chiave.

Il tracker è stato in grado di seguire la chiave in vari condizioni di luce, ma sono emersi due problemi rilevanti. Prima di tutto non ogni set di chiavi presenta delle linee diritte parallele, dato che alcune presentano un effetto di bombatura e hanno bordi curvi. Un altro aspetto da considerare è che le chiavi appese ai pioli non sono fissate saldamente ma possono oscillare leggermente, comportando frequenti cambiamenti di riflessi su di esse. Entrambi i problemi influenzano la robustezza del tracker, che in questi casi può perdere facilmente il tracking del moving edge.

Molto più stabile si è rivelato essere il **template tracker**. Come spiegato in precedenza, esso permette di stimare la trasformazione tra un template di riferimento e la sua posizione nelle immagini successive. La trasformazione può essere di quattro tipi:

1. *2D translation*: considera esclusivamente lo spostamento sui due assi (X e Y).

$$w(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{x} + \boldsymbol{t} \quad \text{con i parametri } \boldsymbol{p} = (t_x, t_y)$$

2. *2D scale rotation translation* (SRT): descrive il fattore di scala, la rotazione sull'asse Z e la traslazione 2D del punto (1).

$$w(\boldsymbol{x}, \boldsymbol{p}) = (1 + s)\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t} \quad \text{con i parametri } \boldsymbol{p} = (s, \theta, t_x, t_y)$$

3. *Affine displacement*: trasformazione che preserva punti, linee rette e piani.

$$w(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{t} \quad \text{con i parametri } \boldsymbol{p} = (a_0 ... a_3, t_x, t_y),$$

$$\boldsymbol{A} = \begin{pmatrix} 1 + a_0 & a_2 \\ a_1 & 1 + a_3 \end{pmatrix}, \ \boldsymbol{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

4. *Homography*: rappresenta tutte le possibili trasformazioni, lineari e prospettiche.

$$w(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{H}\boldsymbol{x} \quad \text{con i parametri } \boldsymbol{p} = (p_0 ... p_7), \ \boldsymbol{H} = \begin{pmatrix} 1 + p_0 & p_3 & p_6 \\ p_1 & 1 + p_4 & p_7 \\ p_2 & p_5 & 1.0 \end{pmatrix}$$

Il `vpTemplateTracker` implementato in ViSP richiede di definire il template di riferimento con uno o più **triangoli complanari**. Per il task della challenge la testa della chiave è stata inizialmente racchiusa dentro un singolo triangolo; in seguito sarà possibile utilizzare direttamente il quadrato restituito dal classificatore.

Per creare il template tracker è necessario specificare alcuni **parametri**:

- il tipo di *trasformazione*;
- la *funzione di similarità* e la sua modalità (ad esempio *SSD Forward Additional*);
- quanto le immagini devono essere ridotte (*subsampled*);
- il *guadagno* (*gain*) usato nel loop di ottimizzazione;
- il numero massimo di *iterazioni* per il loop di ottimizzazione.

Questi ultimi tre parametri in particolare influenzano le **performance** del tracker in termini di tempo e accuratezza. Infatti la fase di tracking si basa su un algoritmo iterativo che minimizza una funzione di costo: riducendo il numero di iterazioni, il programma risulta più veloce, rischiando tuttavia di cadere in un minimo locale della funzione. Per quanto riguarda il subsampling, ridurre il numero di pixel considerati porta di conseguenza ad un minore tempo di elaborazione, riducendo tuttavia l'efficienza. Per ottenere le performance desiderate è quindi necessario un preciso tuning dei parametri, trovando un buon compromesso tra velocità e precisione.

L'approccio scelto, dopo alcuni test, è stato il seguente: un template tracker che utilizza la funzione di similarità *SSD (Inverse Compositional)* e la trasformazione *affine*.

Ricevendo le immagini da una webcam il programma sviluppato riesce a lavorare *in tempo reale* senza effettuare un subsampling delle immagini; tuttavia, passando a telecamere di risoluzione maggiore, questo potrebbe rivelarsi necessario per garantire la stessa velocità di processing. La soluzione adottata si è mostrata piuttosto robusta a variazioni di scena, come i riflessi di luce sulle chiavi. Muovendo la telecamera alla velocità presumibile del braccio robotico questo tracker riesce a seguire correttamente la chiave in tempo reale (Figura 5.9).

(a) (b) (c)

FIGURA 5.9: Tracking della chiave mediante il template tracker sviluppato.

5.3.2 Visual Servoing

Per la MBZIRC Challenge si era pensato di utilizzare il **visual servoing** per diversi task: per esempio, portare l'end-effector del braccio manipolatore, e di conseguenza il gripper, davanti alla chiave identificata. Un ulteriore compito era di applicare il visual servoing per posizionare il gripper davanti alla valvola, prima dell'inserzione della chiave sulla stessa. In realtà l'ipotesi di utilizzare il visual servoing per muovere il robot è stata successivamente abbandonata, in favore della più semplice e diretta tecnica della sovrapposizione delle `tf`, descritta nel capitolo 4.

Esaminando le varie opzioni disponibili in ViSP, si è scelto di testare la seguente configurazione. Per configurazione del robot il feedback visivo era di tipo **eye-in-hand**, data la posizione della stereo camera sull'end-effector del braccio manipolatore. Considerato che l'oggetto da raggiungere viene visto sempre frontalmente, e che il tracking è realizzato in 2D, si è provato un visual servoing **image-based**.

L'idea è stata di sfruttare le informazioni fornite dal template tracker per implementare il visual servoing. In particolare si sono considerati i **tre vertici** del triangolo che definisce il template di riferimento come visual feature per il servoing. Sviluppando ulteriormente il programma di tracking si sono ottenuti per ogni frame le coordinate immagine dei tre vertici. Queste hanno rappresentato il **set corrente** delle visual feature **s**. Il set desiderato **s***, invece, è stato imposto manualmente con delle coordinate *target* sul piano immagine, rappresentanti sempre un triangolo. Per esempio, se si vuole posizionare la stereo camera davanti alla chiave selezionata, si deve assegnare al set **s*** le coordinate di un triangolo al centro del piano immagine (Figura 5.10).

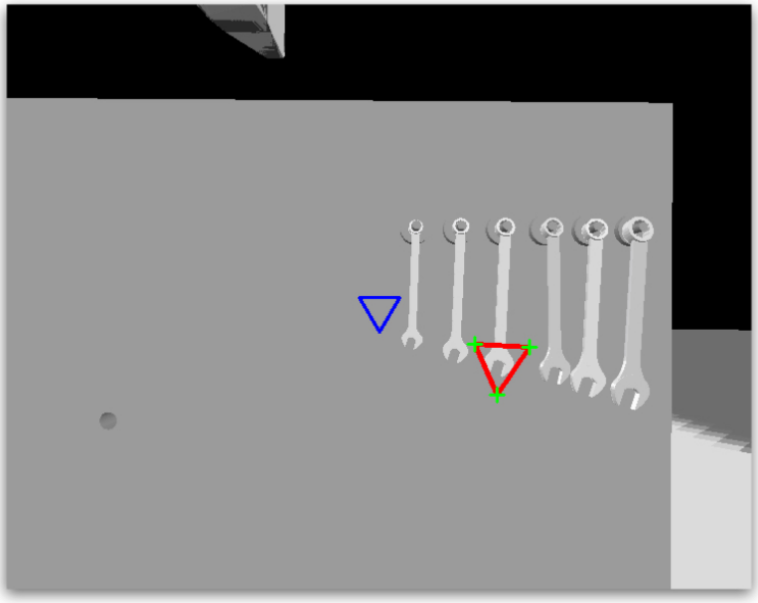


FIGURA 5.10: In blu il set desiderato **s*** e in rosso il set corrente **s**.

Le seguenti parti di codice mostrano come il task del visual servoing si integri con le informazioni fornite dal template tracker. In Figura 5.11 si può osservare come i vertici del triangolo del template tracker per ogni frame vengano convertiti nelle feature del set corrente **s**.

```
// Get the triangle corners
triangle.getCorners(current_points);

// Convert them in visual servo features
for(int j=0; j<current_points.size(); j++)
    vpFeatureBuilder::create(p[j], cam, current_points[j]);
```

FIGURA 5.11: I vertici del triangolo del tracker vengono convertiti in feature correnti per il task del visual servoing.

In conclusione viene computata la **legge di controllo** e il vettore restituito in output contiene le sei velocità (*lineari* e *angolari*) che devono essere applicate sul frame della stereo camera per raggiungere il bersaglio (Figura 5.12).

```
// Compute the visual servoing control law:
// "v" is a vector of 6 velocities in the camera frame
vpColVector v = task.computeControlLaw();
```

FIGURA 5.12: Calcolo della legge di controllo che restituisce il vettore di velocità per raggiungere il target.

La legge di controllo utilizzata è la `EYEINHAND_CAMERA`[29] di ViSP:

$$\boldsymbol{v_c} = -\lambda \, L_x^+ \, \boldsymbol{e},$$

dove $\boldsymbol{v_c}$ è il vettore di velocità da applicare alla stereo camera, λ è il guadagno (*gain*), L_x^+ è la matrice di interazione ed $\boldsymbol{e}$ è l'errore $(\boldsymbol{s} - \boldsymbol{s}^*)$ da minimizzare.

5.4 Test in simulazione

5.4.1 Object Tracking

Testing in Gazebo

Dopo aver provato il template tracker in reale, tramite una webcam, si è scelto di testarlo anche in simulazione. L'ambiente di simulazione usato inizialmente è stato **Gazebo**[1], dove è stata ricostruita la scena della challenge con i vari modelli richiesti: la base mobile, il braccio manipolatore UR5, il pannello con la valvola e le chiavi appese.

Per prima cosa è stata aggiunta al **modello URDF** del robot una telecamera stereo all'estremità del braccio manipolatore, sotto al gripper. L'URDF (*Universal Robotic Description Format*) è un formato di file XML utilizzato in ROS per descrivere i vari elementi di un robot. Per semplificare la descrizione e per ridurre la quantità del codice, *Xacro* (XML Macros) viene spesso usato quando si lavora con i file URDF. Xacro è un linguaggio XML che permette di costruire file XML più corti e leggibili usando delle

[1]Gazebo: http://gazebosim.org

macro che si espandono in espressioni XML più lunghe. In questo modo Xacro permette una maggiore modularità e riutilizzo del codice quando si definisce un modello URDF.

Aggiunto il modello di stereo camera, è stato collegato il **plugin** correlato di interfacciamento tra ROS e Gazebo: `libgazebo_ros_multicamera_so`[11]. Questi plugin forniscono supporto per comunicare con un nodo ROS, in particolare per l'output dei sensori e l'input motorio. Il plugin della stereo camera permette di pubblicare le informazioni della camera (*camera_info*) e le immagini attraverso i topic specificati (Figura 5.13).

```
<plugin name="stereo_camera_controller" filename="libgazebo_ros_multicamera.so">
  <alwaysOn>true</alwaysOn>
  <updateRate>0.0</updateRate>
  <cameraName>multisense_sl/camera</cameraName>
  <imageTopicName>image_raw</imageTopicName>
  <cameraInfoTopicName>camera_info</cameraInfoTopicName>
  <frameName>left_camera_optical_frame</frameName>
  <rightFrameName>right_camera_optical_frame</rightFrameName>
  <hackBaseline>0.09</hackBaseline>
  <distortionK1>0.0</distortionK1>
  <distortionK2>0.0</distortionK2>
  <distortionK3>0.0</distortionK3>
  <distortionT1>0.0</distortionT1>
  <distortionT2>0.0</distortionT2>
</plugin>
```

FIGURA 5.13: Plugin di interfacciamento ROS-Gazebo per la stereo camera.

A questo punto, si è inserito il template tracker sviluppato in precedenza in un nodo ROS, in grado di acquisire le immagini tramite **topic**, provenienti indifferentemente da una telecamera virtuale o reale. Si è quindi creato un semplice script per muovere il braccio manipolatore lungo il pannello, in modo da testare il tracking della chiave.

Come previsto, i risultati sono stati migliori in simulazione che nel reale, a causa di una visione più semplice con meno dettagli e variazioni di luce (Figura 5.14).

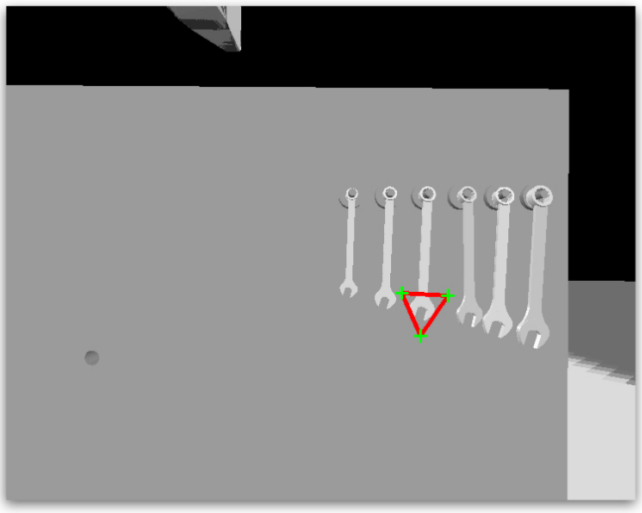


FIGURA 5.14: Test di tracking della chiave in Gazebo.

Testing in V-Rep

A causa dell'insorgere di numerosi problemi nell'usare Gazebo, il team ha deciso di passare a V-Rep[2], un programma di simulazione più sofisticato. Per effettuare il porting dell'applicazione di tracking è stato necessario aggiungere il modello della stereo camera al braccio del robot (Figura 5.15), e renderlo in grado di pubblicare immagini attraverso i topic ROS.

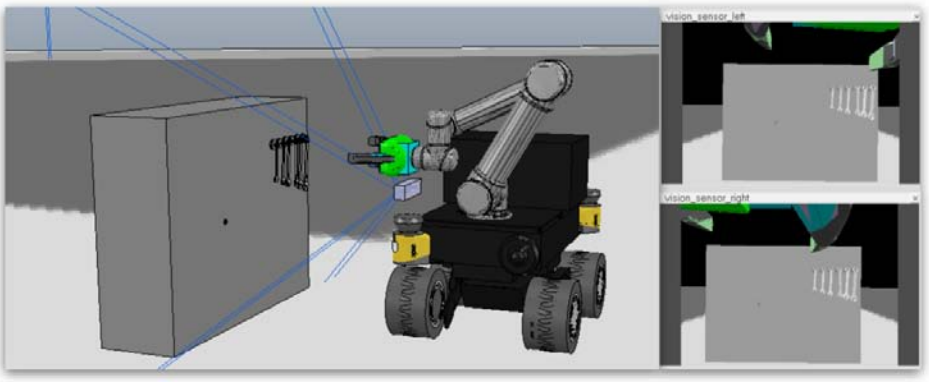


FIGURA 5.15: Modello del robot in V-Rep e visione dell'immagine sinistra e destra.

Per permettere la comunicazione con i nodi ROS, V-Rep fornisce una utile interfaccia attivata da un plugin apposito. V-Rep in questo modo può agire come un vero e proprio nodo ROS, con cui altri nodi possono comunicare tramite servizi, *publisher* e *subscriber*. E' stato così aggiunto uno *script child* all'oggetto della stereo camera che crea un publisher (Figura 5.16). Questo publisher riesce a spedire le immagini, ricevute dal sensore virtuale, attraverso il topic selezionato.

```
-- Retrive the handle of the vision sensor we wish to stream:
visionSensorHandleLeft=simGetObjectHandle('vision_sensor_left')

-- Now enable topic publishing and streaming of the vision sensor's data:
topicName=simExtROS_enablePublisher('visionSensorDataLeft', 1,
      simros_strmcmd_get_vision_sensor_image, visionSensorHandleLeft, 0, '')
```

FIGURA 5.16: Script per la telecamera sinistra che crea il publisher e spedisce le immagini tramite topic ROS.

Come in Gazebo, il programma di tracking sviluppato è riuscito ad operare correttamente in V-Rep ricevendo immagini dalla camera virtuale.

[2]V-Rep: http://www.coppeliarobotics.com

5.4.2 Visual Servoing

Testing in V-Rep (prima modalità)

Il programma di visual servoing sviluppato è stato testato nell'ambiente di simulazione V-REP in due modalità.

Innanzitutto, si è sfruttato un nodo ROS in grado di controllare i motori delle ruote della base mobile virtuale. L'obbiettivo era muovere il robot lungo il pannello utilizzando le informazioni di velocità fornite dal visual servoing. Per fare ciò, il nodo ROS che controlla le ruote riceve il vettore delle velocità attraverso un topic dal programma di visual tracking-servoing, e dopo alcuni aggiustamenti lo inoltra attraverso un altro topic al controller V-REP delle ruote (Figura 5.17).

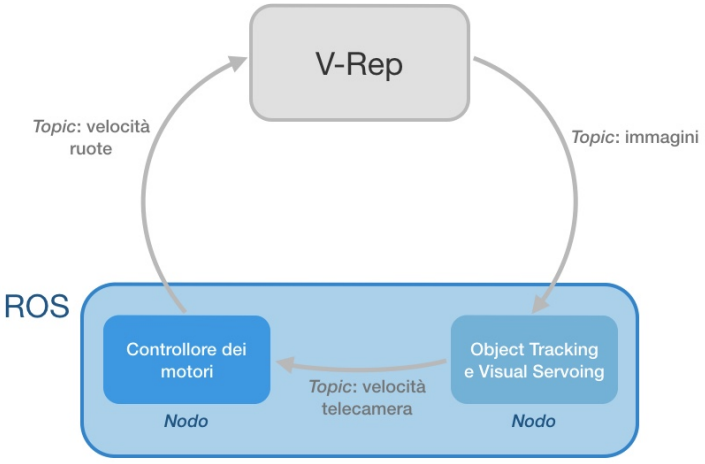


FIGURA 5.17: Schema del primo test di simulazione del visual servoing in V-Rep.

Partita la simulazione la stereo camera virtuale invia le immagini al nodo ROS che esegue il template tracking e il visual servoing, che pubblica i risultati della velocità da applicare. Questi sono letti dal nodo ROS che controlla il motore della base mobile, che a sua volta inoltra un comando *forward* o *backward* al controllore V-Rep delle ruote.

In questo modo è stato possibile effettuare un primo test di visual servoing, nonostante fosse solo sull'asse parallelo al pannello: il robot in ogni caso si è mosso nella giusta direzione e si è posizionato davanti alla chiave scelta.

Testing in V-Rep (seconda modalità)

Un modo più preciso di simulare il visual servoing in V-Rep è stato sviluppato in seguito: si è così cercato di muovere il braccio robotico UR5 invece della base mobile. Il problema principale riguarda il calcolo delle velocità da applicare ad ogni giuntura del braccio, avendo solo le velocità desiderate dell'end-effector. Un esaustivo esame della cinematica inversa era al di là del progetto di questa tesi, per cui si è scelto di utilizzare lo strumento integrato da V-REP per integrare questa funzionalità.

V-Rep permette di specificare un **task di Inverse Kinematics** (IK) aggiungendo i link della catena cinematica, descritta da un *tip dummy* e un *base object*. In seguito è necessario creare un **target dummy**, e collegarlo al tip dummy. Così facendo il tip è costretto a seguire il target, muovendo di conseguenza tutta la catena cinematica. Si è così formata una catena con i vari giunti dell'UR5, impostando la modalità IK. Il tip dummy è stato poi posizionato al centro dell'end-effector, mentre il target dummy è stato aggiunto vicino al robot e collegato al tip con un *tip-target link* (Figura 5.18).

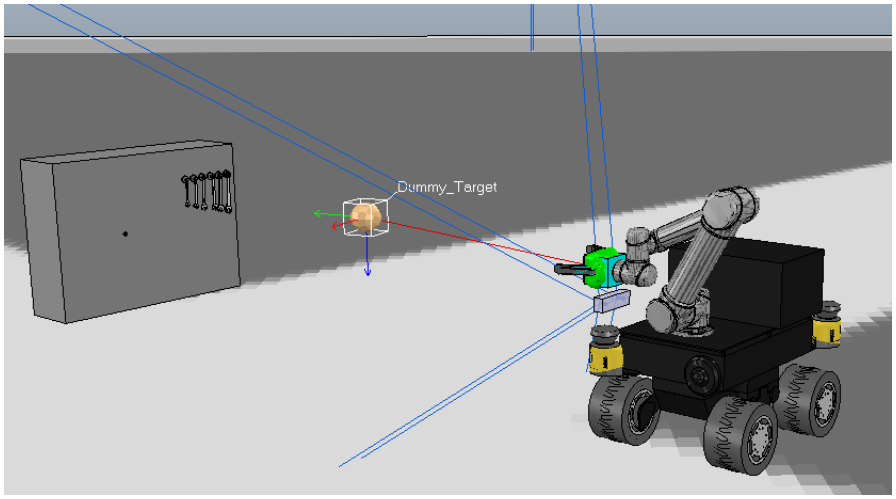
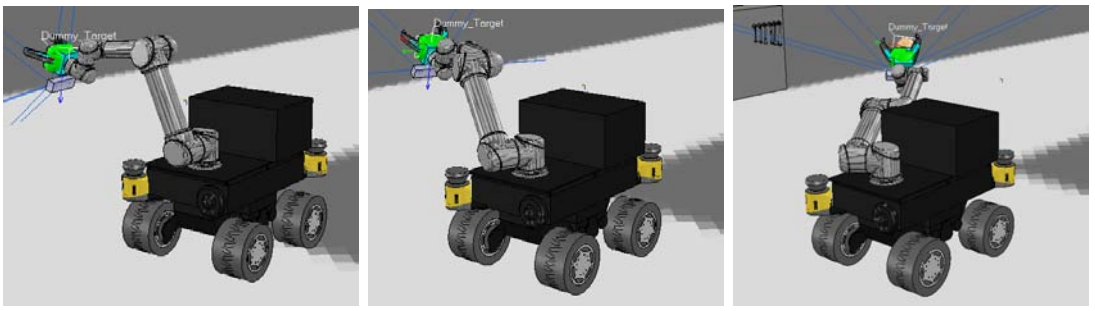


FIGURA 5.18: Test con catena cinematica e tip-target dummy in V-Rep.

Come si può vedere nella Figura 5.19, quando il dummy target viene mosso manualmente nello spazio tutto il braccio del robot si muove di conseguenza, seguendo i suoi movimenti e cercando di raggiungerlo. Tramite script, è possibile infine assegnare le velocità restituite dal programma di visual servoing al dummy target, facendo così muovere l'intero braccio nella posizione desiderata.



(a) (b) (c)

FIGURA 5.19: Simulazione del visual servoing con catena cinematica e tip-target dummy in V-Rep.

5.5 Risultati nel mondo reale

Come detto in precedenza, il visual servoing non è stato infine utilizzato per la Challenge MBZIRC. Per questo motivo sono stati effettuati solo limitati test con l'hardware reale, non in simulazione. Tuttavia i risultati preliminari ottenuti sono stati interessanti, e invitano ad un possibile sviluppo futuro.

L'hardware utilizzato è stato il braccio manipolatore UR10, sostanzialmente identico all'UR5 per funzionalità ma di dimensioni maggiori, e la stereo camera Bumblebee 2, fissata all'end-effector. Davanti ad essa si è posto un pannello con un insieme di chiavi appese, per simulare l'ambiente della challenge. Per testare il programma di visual servoing nella realtà sono state richieste due implementazioni. La prima è stata di passare dall'input utente del template tracker, che consiste nel selezionare i tre vertici del triangolo racchiudente la chiave, ad un input automatico. Per fare ciò si è utilizzato l'object detector sviluppato da Matteo Munaro (Ph.D. dello IAS Lab), che localizza le chiavi e sceglie quella da prendere. Tramite un topic ROS si sono ricevute da questo nodo di detection le coordinate del bounding box attorno alla testa della chiave. In questo modo il nodo di object tracking è in grado di creare autonomamente il template di riferimento. La seconda parte è stato di utilizzare la Robot Movement Interface[3] per controllare il movimento dell'UR10. Questa interfaccia fornisce un'implementazione ROS per trasformare comandi human-readable in comandi di basso livello da inviare al robot. Assieme a questa interfaccia si sono usati anche i driver ROS della UniversalRobotic[4].

[3]Robot Movement Interface: https://github.com/ros-industrial/robot_movement_interface
[4]Driver ROS UR: http://wiki.ros.org/universal_robot

I seguenti frammenti di codice mostrano come sono stati creati ed inviati i comandi di velocità al braccio manipolatore. Nel codice in Figura 5.20 viene creato il publisher che invia i comandi al controller dell'UR10, e inizializzata la lista dei comandi.

```
//Publish commands to the robot
ros::Publisher command_pub =
        n.advertise<
                ipa325_robot_movement_action_v2::ipa325_robot_movement_v2_command_list>(
                "ipa325_robot_command_list", 5, true);

ROS_INFO("UR10 Test ready.");

//Initialization of the commands for the robot
ipa325_robot_movement_action_v2::ipa325_robot_movement_v2_command_list trajectory;
```

FIGURA 5.20: Codice per la creazione del publisher e l'inizializzazione della lista dei comandi.

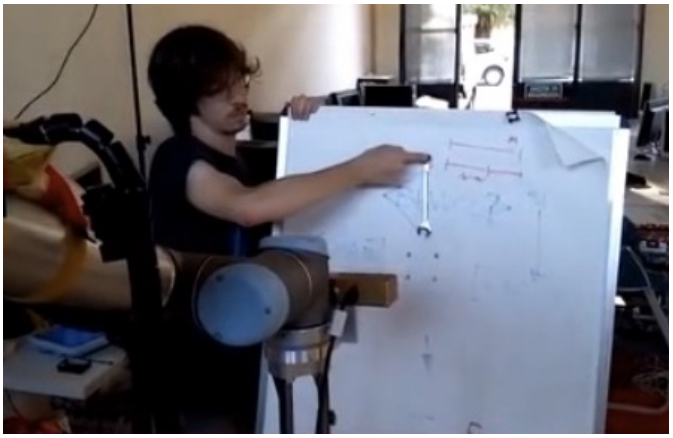
Nel codice della Figura 5.21 viene invece mostrato come creare un comando di velocità, in questo caso con tutte velocità nulle, e come pubblicarlo. E' possibile scegliere un *id*, una tipologia (nell'esempio comando in *velocità cartesiana*), e le *unità di misura* (nell'esempio di velocità e di accelerazione).

```
//First command
ipa325_robot_movement_action_v2::ipa325_robot_movement_v2_command c0;
c0.command_id = 0;
c0.command_type = "CARTESIAN_SPEED";
float velocityNull[] = { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
c0.velocity.assign(velocityNull, velocityNull + 6);
c0.velocity_type = "RAD/S";
c0.acceleration_type = "RAD/S^2";
float acceleration[] = { 0.01f };
c0.acceleration.assign(acceleration, acceleration + 1);

ipa325_robot_movement_action_v2::ipa325_robot_movement_v2_command commands[] = { c0 };
trajectory.commands.assign(commands, commands + 1);
trajectory.replace_previous_commands = true;

//Publish the command
command_pub.publish(trajectory);
```

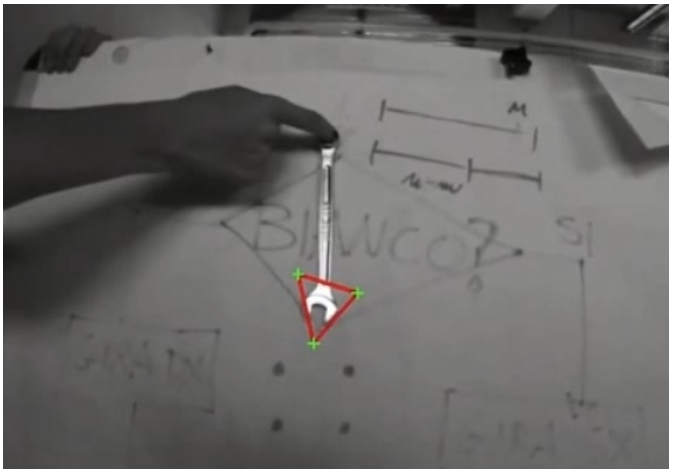
FIGURA 5.21: Codice per la creazione e l'invio di un comando di velocità nulle.

In questo modo quando l'algoritmo di visual servoing calcola le velocità da applicare viene generato un nuovo comando. Per testare in sicurezza le velocità sono state ridotte prima di essere inviate al braccio manipolatore, così da evitare danni al braccio robotico.

I risultati ottenuti nei test effettuati sono stati promettenti, con l'UR10 che si muove correttamente davanti al pannello seguendo una chiave che viene mossa manualmente (Figura 5.22).



(a)



(b)

FIGURA 5.22: Test dell'applicazione di visual servoing nel mondo reale: in (a) una visione esterna dell'end effector che si muove davanti alla chiave, in (b) la visione della telecamera del robot, con il tracking della chiave.

Capitolo 6

Conclusioni

In conclusione, il lavoro di questa tesi ha coperto i seguenti ambiti:

1. *Object Detection*: sono stati creati dei boosted cascade classifier per localizzare le chiavi e la valvola. Anche in condizioni ambientali di luce intensa, riflessi e ombre, questi classificatori hanno restituito ottimi risultati, riuscendo a identificare gli oggetti con una precisione maggiore del 90%. Addirittura in gara il classificatore delle chiavi ha eseguito 225 detection corrette su 225.

2. *3D Pose Estimation*: sono stati sviluppati vari programmi in grado di determinare la posizione e l'orientazione tridimensionale degli oggetti. Si sono quindi creati dei frame di coordinate al centro della chiave per il grasping, al centro della sua apertura (con la corretta orientazione) per l'inserzione, e al centro della valvola, seguendo l'orientazione dello stelo.

3. *Object Tracking e Visual Servoing*: è stato implementato un programma in grado di tracciare e seguire una chiave in una sequenza di immagini o video, fornendo il feedback visivo per muovere il braccio manipolatore di conseguenza. Grazie al visual servoing infatti è possibile posizionare il robot davanti al target desiderato.

Benché la tesi sia stata incentrata sulla challenge MBZIRC, il lavoro svolto permette applicazioni più generali, come la detection di oggetti riflettenti, la stima della posizione mediante stereo visione e il pacchetto `tf`, l'individuazione di forme geometriche semplici con la trasformata di Hough, e il movimento di un sistema robotico in base ai feedback visivi.

In conclusione di questo percorso, la squadra Desert Lion di cui ho fatto parte ha ottenuto il terzo posto nella Grand Challenge MBZIRC, il 18 Marzo 2017 ad Abu Dhabi (Figura 6.1).

FIGURA 6.1: Desert Lion team: 3^{rd} place in the Grand Challenge MBZIRC.

Bibliografia

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, June 2008.

[2] J. Brownlee. Classification accuracy is not enough: More performance measures you can use, 2010. URL `http://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/`.

[3] François Chaumette, Ezio Malis, and Sylvie Boudet. 2d 1/2 visual servoing with respect to a planar object, 1997.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, September 1995.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 2005.

[6] A. Dame and E. Marchand. Accurate real-time tracking using mutual information. *IEEE Int. Symp. on Mixed and Augmented Reality*, October 2010.

[7] César de Souza. Haar-feature object detection in c-sharp, 2014. URL `https://www.codeproject.com/Articles/441226/Haar-feature-Object-Detection-in-Csharp`.

[8] Boston Dynamics. Cheetah - fastest legged robot. URL `http://www.bostondynamics.com/robot_cheetah.html`.

[9] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, September 2010.

[10] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. 1995.

[11] Gazebo. Tutorial: Using gazebo plugins with ros. URL `http://gazebosim.org/tutorials?tut=ros_gzplugins`.

[12] ISO. Assembly tools for screws and nuts – combination wrenches – lengths of wrenches and maximum thickness of heads, 2015. URL `https://www.iso.org/standard/62551.html`.

[13] A. Kaehler and G. Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library.* O'Reilly Media, 2016. URL `https://books.google.it/books?id=LPm3DQAAQBAJ`.

[14] Kurt Konolige. Small vision systems: Hardware and implementation. *Robotics Research: The Eighth International Symposium*, 1998.

[15] Point Cloud Library. Pcl::segmentation, 2011. URL `http://www.pointclouds.org/assets/iros2011/segmentation.pdf`.

[16] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, November 2004.

[17] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, 1981.

[18] S. Mallick. Image recognition and object detection, 2016. URL `http://www.learnopencv.com/image-recognition-and-object-detection-part1/`.

[19] E. Marchand and F. Spindler. ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *Robotics & Automation Magazine, IEEE*, 2005.

[20] S. Naotoshi. Tutorial: Opencv haartraining (rapid object detection with a cascade of boosted classifiers based on haar-like features), 2006. URL `http://note.sonots.com/SciSoftware/haartraining.html`.

[21] Timo Ojala, Matti Pietikainen, and David Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. 1994.

[22] OpenCV. Hough line transform. URL `http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html`.

[23] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL), May 9-13 2011.

[24] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. 1998.

[25] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. *International Journal of Computer Vision*, 1991.

[26] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. 2001.

[27] ViSP. Klt tracker module overview, . URL `https://visp.inria.fr/klt/`.

[28] ViSP. Blob tracker module overview, . URL `https://visp.inria.fr/blob/`.

[29] ViSP. Visp: vpservo class reference, . URL `http://visp-doc.inria.fr/doxygen/visp-2.8.0/classvpServo.html`.

[30] ViSP. Markerless 3d model-based tracker module overview, . URL `https://visp.inria.fr/mbt/`.

[31] ViSP. Moving-edge tracker module overview, . URL `https://visp.inria.fr/moving-edges/`.

[32] ViSP. Template tracker module overview, . URL `https://visp.inria.fr/template-tracking/`.

[33] ViSP. Tutorial: Template tracking, . URL `http://visp-doc.inria.fr/doxygen/visp-daily/tutorial-tracking-tt.html`.

[34] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vision Comput.*, February 1990.