UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un sistema di elaborazione di immagini per la ricostruzione 3D di frutti mediante Structure from Motion

Design and Development of a image processing system to generate 3D models of fruits using Structure from Motion

Relatore:
Prof. Mancini Adriano

Laureando:
Alessio
Brugiavini

Anno Accademico 2020/2021

Abstract

Una nuvola di punti o *point cloud* è,come suggerisce il nome, un insieme di punti che possiede attributi come colore e posizione in un sistema di riferimento 3D. Il campo di applicazione di questi software è estremamente vasto in quanto tale tecnologia può agevolare lavori di progettazione e di analisi, del tipo effettuare misure dettagliate di un modello 3D istantaneamente oppure osservare l'evoluzione di un sistema biologico. Sono presenti sul mercato strumenti hardware, e applicazioni software, specializzate nell' effettuare una costruzione tridimensionale basata su *Point Cloud Library (PCL)*: utilizzando un *dataset* di foto ottenute in precedenza, oppure tramite l'utilizzo di dispositivi di acquisizione come telecamere o sistemi di telerilevamento avanzati (come ad esempio *LIDAR*). Nell'ambito di questa tesi si è scelto di effettuare un confronto tra diversi software, hardware presenti sul mercato e una ricostruzione di un modello 3D mediante OpenCV implementata in Python.

Indice

Capitolo 1

Introduzione

L'obiettivo di questo elaborato è di progettare e sviluppare un software in Python in grado di generare il modello tridimensionale di frutti e visualizzarlo in un ambiente specifico chiamato Potree[1], in modo da semplificare l'analisi della struttura di come evolvono i frutti nel tempo. Nella creazione di questo software, avvenuta assieme al collega Cattani Lorenzo, si è deciso di implementare diverse funzionalità che permetteranno all'utente di stabilire i propri parametri qualitativi della ricostruzione. Inoltre sono stati utilizzati diversi strumenti per la realizzazione dei modelli tridimensionali in modo da confrontarne i risultati e rendersi conto delle possibili implementazioni future. Il progetto si basa principalmente sull'utilizzo di una libreria grafica OpenCV[2] specifica per progetti inerenti alla computer *vision* e *image processing*.

1.1 Struttura della tesi

La tesi viene strutturata come segue:

- nel secondo capitolo vengono introdotti gli strumenti software e hardware utilizzati durante lo sviluppo del progetto.
- il terzo capitolo riguarda la procedura di acquisizione delle immagini: qui sono presentate le buone pratiche per la corretta configurazione della fotocamera e dell'ambiente.
- nel quarto capitolo vi è un introduzione teorica riguardante il calcolo della matrice di camera (ovvero una matrice contenente dei parametri fondamentali) e dei coefficienti di distorsione, ovvero dei parametri utili per correggere la distorsione dell'immagine che poi verrà utilizzata per la ricostruzione 3D
- nel quinto capitolo è introdotta e spiegata la funzione , la quale ha il compito di prendere in ingresso i parametri calcolati nel capitolo precedente e di applicarli all'immagine scelta per la ricostruzione 3D
- nel sesto capitolo vi è una introduzione teorica riguardante il calcolo della depthmap, ovvero un'immagine in scala di grigi, dove ad ogni tonalità

di grigio è associata la distanza da un riferimento. Segue la procedura di elaborazione delle immagini tramite Python.

- nel settimo capitolo sono analizzati e utilizzati diversi filtri da applicare alla depth map, in modo da migliorare la costruzione della nuvola di punti , che avviene nel sesto capitolo.
- nell'ottavo capitolo è introdotta e realizzata la nuvola di punti attraverso l'utilizzo di una depth map. A seguire vi è illustrata la procedura di salvataggio della nuvola di punti prodotta. Il capitolo si conclude con la generazione della point cloud ottenuta da software professionali.
- nel nono e ultimo capitolo si espongono le conclusioni del progetto, valutando i risultati ottenuti.

Capitolo 2

Strumenti e metodi

In questa tesi sono stati utilizzati diversi strumenti, sia software (SW) che hardware (HW). La licenza legata a tali strumenti non è quindi sempre gratuita o open source, anche a causa della complessità necessaria alla loro realizzazione. Il confronto effettuato nel capitolo sette tiene conto anche di questi fattori. In seguito sono riportati i principali strumenti utilizzati:

- l'acquisizione delle foto è avvenuta tramite una Reflex Nikon D3300.
- dalla fase di creazione della *depth map*, sono stati utilizzati i SW Matlab[3] e Pycharm.
- la fase di generazione e visualizzazione del *point cloud* è avvenuta tramite Kinect, Metashape, e dalla nostra applicazione in Python, che una volta generati i file di salvataggio, provvederà a caricarlo in un `webserver` tramite Xampp[4], la cui visualizzazione è realizzata tramite Potree[1].

2.1 Python

Il linguaggio di sviluppo scelto è Python: un linguaggio polivalente, divenuto molto popolare per la sua semplicità nella leggibilità del codice. Rispetto agli altri linguaggi come C/C++ risulta più lento nella computazione, quindi la gestione dei controlli interni al codice richiede più tempo in fase di runtime, ma assicura un'esperienza in fase di sviluppo molto più diretta e semplice.

Figura 2.1: Logo Pycharm: ambiente di sviluppo utilizzato[5]

2.1.1 OpenCV e Numpy

In Python ci sono due librerie open source adatte al nostro scopo: la prima è OpenCV (Open Source computer Vision Library), sviluppata dalla divisione

russa della Intel nel 1999. OpenCV è legata all'ambito della visione artificiale real time dedicata esclusivamente alla creazione e manipolazione di immagini e oggetti tridimensionali, che risulta particolarmente efficiente se combinata con Numpy[6], libreria estremamente ottimizzata per operazioni numeriche, dato che le immagini all'interno del nostro linguaggio sono rappresentate tramite matrici multidimensionali. Quest'ultima, combinata con la precedente, ci offre uno strumento efficace per la risoluzione di problemi di *computer vision*.

2.1.2 PLY e LAS

Il risultato dell'elaborato sarà quello di produrre modelli tridimensionali, in particolare di frutti e ortaggi. Per usufruire di una eventuale portabilità e compatibilità con i SW presenti sul mercato, abbiamo scelto due tra i formati più diffusi degli elaborati tridimensionali: il Polygon File Format[7] e il LAS[8]. Il Polygon File Format o Ply è molto diffuso per la conservazione di informazioni provenienti dagli scanner 3D. La ricostruzione di questi file parte dal salvataggio di vertici di poligoni bidimensionali con caratteristiche specifiche come colore e trasparenza. I dati possono essere salvati nel formato ASCII o binario, basta specificarlo in fase di inizializzazione nell'header.

Il LAS[8] è il formato più diffuso per la manipolazione di *point cloud* di grandi dimensioni (anche centinaia di GB) e non reagiscono negativamente alla compressione. Il LAS[8] file è stato creato per ottimizzare la gestione di file ottenuti tramite LiDAR[9] (tecnica di telerilevamento tramite impulsi laser) utilizzata per rilevamenti geografici di grandi dimensioni ottenuti tramite satelliti o laser. È stata scelta a questa estensione per le possibili evoluzioni future del software. La struttura del file è organizzata dall'header, che stabilisce i tipi di record utilizzati dalla versione del LAS[8] in uso. Successivamente va stabilita la dimensione delle variabili considerate e le informazioni associate ad ogni vertice. La versione da noi utilizzata è la 1.2[10],[11], mentre la versione dell'header è la 2, perché le altre non permettevano di salvare informazioni utili per il progetto oppure possedevano parametri non necessari che avrebbero appesantito l'esecuzione del SW.

2.2 Potree

Per evitare possibili problemi di compatibilità tra il Sistema Operativo e visualizzatore locale si è deciso di utilizzare Potree[1], SW gratuito che permette di caricare e osservare il *point cloud* dal proprio browser anche in formato LAS[8].

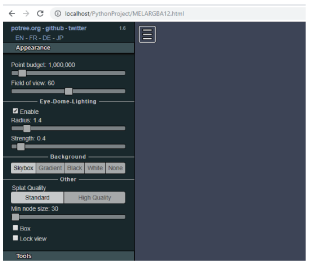

Figura 2.2: Particolare dell'ambiente grafico Potree[1]

2.3 Xampp

La visualizzazione avviene tramite server locale creato nel nostro caso tramite Xampp[4], SW estremamente semplice da configurare e utilizzare.

Figura 2.3: Logo di Xampp[4]

2.4 Bitbucket

Sito di hosting di Version Control System[12] utilizzato per la cooperazione durante lo sviluppo del progetto tra i membri del team. Si appoggia a Git e ci permette di evitare eventuali perdite di dati o l'aggiornamento simultaneo di script.

Figura 2.4: Logo Bitbucket[12]

2.5 Agisoft Metashape

La Agisoft[**?**] ha sviluppato questo software di ricostruzione 3D mediante *structure from motion*[13] con l'obiettivo di rendere il più semplice possibile la ricostruzione di modelli ottenibili dall'elaborazione di foto arbitrarie basandosi su algoritmi avanzati di allineamento e rendering. Questo software è in grado di correggere in maniera limitata gli errori in fase di acquisizione delle foto (come riflessi etc...). La configurazione dei parametri ottimali alla ricostruzione avviene in maniera quasi del tutto automatizzata, e permette di ottenere risultati estremamente accurati. Abbiamo deciso di utilizzare questo SW per effettuare

un confronto finale tra i risultati da noi ottenuti e quelli generati da un SW professionale.

Figura 2.5: Logo PhotoScan[14]

2.6 Matlab

Matlab[3] è un SW creato per l'analisi numerica e statistica che permette di manipolare oggetti matriciali di grandi dimensioni. Tra tutte le funzioni implementabili, a noi è risultata di particolare interesse la modalità di acquisizione della mappa della disparità, che ci permetterà di valutare l'accuratezza di alcuni parametri fondamentali che utilizzeremo nel nostro elaborato.

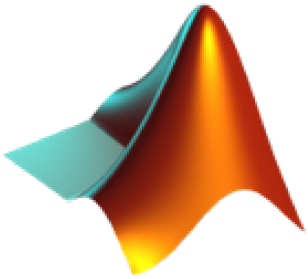

Figura 2.6: Logo Matlab[3]

2.7 Kinect v2

HW sviluppato dalla Microsoft[15] per il *capture* in real time di oggetti e movimenti. In questo progetto il suo utilizzo è limitato all'acquisizione del modello tridimensionale e al suo confronto con i risultati ottenuti.

Figura 2.7: Immagine di Kinect[15]

Capitolo 3

Acquisizione delle foto

Nel processo preliminare di acquisizione delle foto è stata utilizzata una fotocamera Reflex Nikon D3300. Le fotocamere in fase di cattura di un immagine in formato grezzo (JPEG, TIFF e RAW) non applicano le *funzioni correttive* degli errori sistematici, quindi, una volta acquisite, potrebbero essere presenti distorsioni come ad esempio *aberrazione cromatica*, *vignettatura*, *errori trasversali* e *interplanari*[16]. I parametri utilizzati dalle *funzioni correttive* compensano tali distorsioni. È possibile effettuare la correzione della camera manualmente, ma non sarà materia d'analisi in questo elaborato. Il procedimento presentato nei prossimi capitoli, considera il caso in cui si hanno a disposizione tali parametri.

L'acquisizione deve avvenire in condizione di ambiente idoneo. Nella scena la luce deve essere più uniforme possibile: si devono evitare coni di luce, riflessioni, ed effetti di sfocatura. È preferibile ma non indispensabile un *background* omogeneo. L'oggetto deve essere fermo, all'interno di tutte le immagini, e ben visibile, né troppo piccolo, né tanto grande da coprire il *background*. Inoltre è necessario che i punti di acquisizione delle foto e dell'oggetto (visti ortogonalmente al piano), compongano un triangolo, ovvero non devono esserci rotazioni rispetto all'oggetto d'interesse, come rappresentato in figura 3.1. Il Fine ultimo dell'acquisizione di una coppia di foto è quello di poter generare poi una coppia di immagini stereo idonee per l'elaborazione 3D.

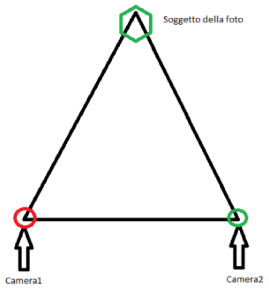

Figura 3.1: Esempio di giusto posizionamento

La qualità delle foto è accettabile se si rispettano le regole appena discusse. Questo è il risultato di un acquisizione in condizioni standard 3.2 6.6.

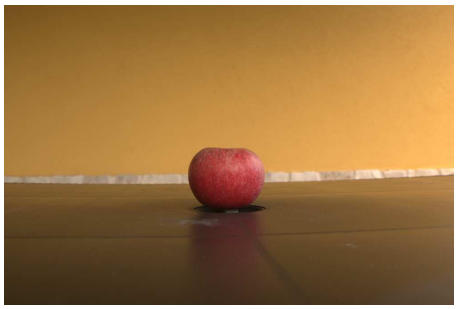

Figura 3.2: Scena ottenuta dalla camera sinistra

Figura 3.3: Scena ottenuta dalla camera destra.

Se queste linee guida non fossero rispettate si potrebbero incorrere in errori di elaborazione derivanti dalla scarsa qualità delle foto in ingresso, che andrebbero poi a ridurre la qualità del lavoro fatto dal software, per cui questo procedimento risulta fondamentale ai fini della riuscita dell'elaborazione finale. Tuttavia ottenere delle ottime foto in condizioni di ambiente standard è effettivamente complicato. Risulta macchinoso eliminare gli effetti di riflessione presenti negli oggetti della scena. Il problema può essere compensato tramite l'uso di appositi box fotografici, in modo da isolare l'oggetto dall'ambiente e diffondere meglio la luce.

3.0.1 Visione stereoscopica

Un sistema di visione stereoscopica necessita di almeno due telecamere, possibilmente identiche. Il sistema a doppia telecamera è caratterizzato da una copia identica di telecamere; la telecamera sinistra viene traslata di una quantità b lungo l'asse X nel sistema di coordinate (X, Y, Z) centrato nella telecamera sinistra. Si ha quindi che l'origine (0, 0, 0) identifica il centro di proiezione della telecamera sinistra, mentre il centro di proiezione della telecamera destra avrà come coordinate (b, 0, 0). In altre parole abbiamo: 1. Due immagini complanari di uguale misura/risoluzione Ncolonne $\times$ Nrighe. 2. Assi ottici paralleli. 3. Un'identica lunghezza focale f. 4. Righe delle due immagini collineari (la riga y di un'immagine è collineare alla riga y della seconda immagine)

La calibrazione dovrà provvedere a valutare accuratamente b e f quando si usa un sistema di Stereo Vision Questi parametri verranno approfonditi nel capitolo successivo.

3.0.2 Setup utilizzato

Per scattare le foto e catturare le immagini di interesse è stata utilizzata una fotocamera reflex *Nikon D3500* , è dotata di un sensore DX, CMOS(*Complementary Metal-Oxide semiconductor*), 23.5 mm x 15.6 mm con una risoluzione di 24.2 megapixel. Le dimensioni delle immagini ottenute sono di 3872 x 2592 pixel. Ogni pixel è grande circa 6.06 micrometri. La distanza focale della lente utilizzata è variabile tra 17 e 55 mm.

Figura 3.4: Fotocamera NikonD3500 vista frontale

Figura 3.5: Fotocamera NikonD3500 vista superiore

Capitolo 4

Calibrazione

La *funzione di calibrazione*[17] è quel processo volto a stimare parametri necessari per correlare i punti del mondo reale (sistema di coordinate mondo) con i punti dell'immagine catturata tramite una fotocamera. Affinché si ottengano parametri migliori, si possono scattare più immagini della stessa scena. I parametri si dividono in due categorie:

- parametri intrinseci: sono parametri che dipendono dalla telecamera (per esempio la lunghezza focale f),
- parametri estrinseci: sono parametri che consentono il cambio di coordinate tra due sistemi di riferimento.

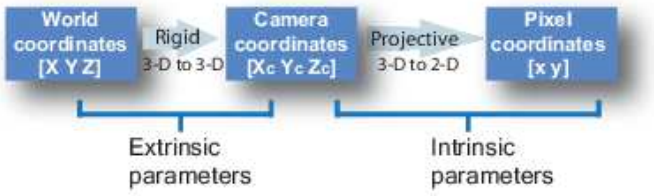

Figura 4.1: parametri della matrice di camera

Ottenere questi parametri viene detto Camera Calibration.

4.0.1 Matrice di Camera

Nell'ambito della *Computer Vision*[18], una *matrice di camera*[17] è una matrice di proiezione 3x4 che descrive la mappatura di una camera prendendo in input una nuvola di punti 3D e restituendo una mappatura di punti 2D in una data immagine. Ipotizzando che x sia la rappresentazione di una nuvola di punti 3D in coordinate omogenee, e che y sia la rappresentazione 2D dell'immagine della camera (un vettore a 3 dimensioni), allora vale la seguente relazione: $y \approx Cx$.

dove C è la matrice di camera[17] e "approx" implica che la parte a destra e sinistra di esso siano uguali ad un prodotto scalare non nullo.

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 4.2: Rappresentazione della *Matrice di camera*

4.0.2 Come calcolare la Matrice di camera

Nel nostro caso, poichè si ha a che fare con *immagini stereo*[19], per trovare questi parametri bisogna utilizzare come campione di riferimento un'immagine con un pattern ben definito, per cui è stata scelta una scacchiera. Il motivo di tale scelta è che le dimensioni dei singoli scacchi che compongono perimetro e area della scacchiera sono ben misurabili e quindi possono fornire una precisa indicazione al software per ricavare i parametri che verranno poi utilizzati.

```
1     def start_calibration(SPESSORE_SCACCO,DIM_ORIZZONTALE_SCACCHIERA,
      DIM_VERTICALE_SCACCHIERA,IMAGES,IMG,IMGCORRECTED)
2
```

come si può notare dal codice, i parametri in ingresso sono proprio i dati geometrici della scacchiera e le immagini da elaborare. Altri dati importanti per la calibrazione della camera sono un insieme di punti 3D del mondo reale e la corrispondente immagine 2D.

```
1     # Arrays to store object points and image points from all the images.
2     objpoints = []  # 3d point in real world space
3     imgpoints = []  # 2d points in image plane.
4
```

L'immagine 2D non crea problemi in quanto può essere facilmente ricavata tramite la scacchiera (sono i punti dove due quadrati toccano i loro vertici nella scacchiera). Mentre invece per l'immagine 3D bisogna considerare le coordinate X,Y,Z e andare a denotare la posizione dei punti.

```
1      chessboard_Dim = (DIM_VERTICALE_SCACCHIERA, DIM_ORIZZONTALE_SCACCHIERA)
2      objp = np.zeros((DIM_ORIZZONTALE_SCACCHIERA * DIM_VERTICALE_SCACCHIERA,
      3), np.float32)
3     objp[:, :2] = np.mgrid[0:DIM_VERTICALE_SCACCHIERA, 0:
      DIM_ORIZZONTALE_SCACCHIERA].T.reshape(-1, 2)
4
```

Ora possiamo prendere in esame l'immagine della scacchiera e calcolare bordi e angoli

```
1       for fname in IMAGES:
2         img = cv2.imread(fname)
3         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
4         print('assegno bordi, pu  impiegare parecchio tempo (fino a 30-40m
      per immagini in 4k)')
5         # Find the chess board corners
6         # ret = true se sono stati trovati e ordinati tutti i bordi
7         ret, corners = cv2.findChessboardCorners(gray, chessboard_Dim)
8         print('CHESS DIM')
9         print(chessboard_Dim)
10        print('bordi assegnati -fuori if')
11
```

Il codice si occupa di prendere ciclicamente in esame le immagini della scacchiera, le ricolora utilizzando un'apposita scala di grigi e poi utilizza la funzione *cv2.findChessboardCorners* per trovare i bordi ed angoli. Se i bordi e gli angoli della scacchiera vengono calcolati e ordinati correttamente si procede all'assegnazione dei suddetti all'immagine di interesse in questo caso se i bordi sono stati trovati correttamente si calibra l'immagine della scacchiera con la funzione *cv2.drawChessboardCorners*.

```
1      if ret == True:
2              print('immagine accettata, ' + fname)
3              objpoints.append(objp)
4              corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1),
       criteria)  # corners2 ha precisione maggiore di corners
5              imgpoints.append(corners2)
6              # disegno i bordi
7              img = cv2.drawChessboardCorners(img, chessboard_Dim, corners2, ret
       )
8              #--- stampo e centro l immagine ---#
9              winname = "immagine bordata"
10             cv2.namedWindow(winname)
11             cv2.moveWindow(winname, 40, 30)  # Move window to (40,30)
12             cv2.imshow(winname, img)
13
14         else:
15             (print(
16                 'NON SONO STATI TROVATI I BORDI DELL IMMAGINE, controllare l
       immagine :' + fname))
17
```

dunque si applicano i bordi e angoli appena calcolati all'immagine di punti in 2D, il risultato finale è l'immagine di interesse con un pattern corretto disegnato su di essa.

Ora ci si può occupare del calcolo vero e proprio della matrice di camera[17] e dei coefficienti di distorsione, per farlo utilizziamo la funzione *cv2.calibrateCamera*, il risultato è visibile nella figura 4.4:

```
1      # objpoints: pt spazio 3d, imgpoints: pt spazio 2d
2      ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
       gray.shape[::-1], None, None)
3
```

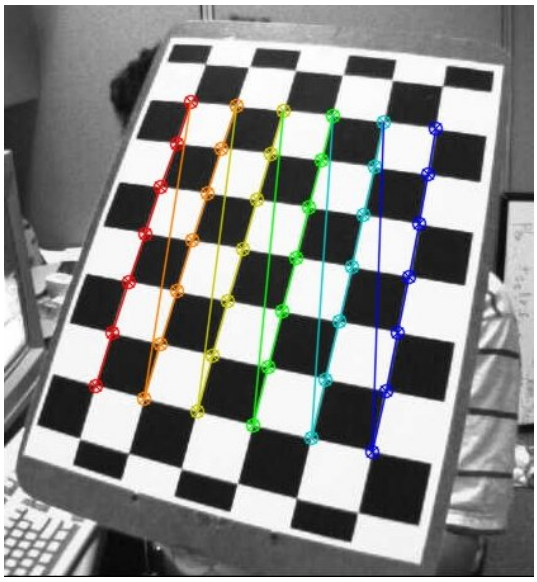

Figura 4.3: scacchiera corretta e calibrata

Capitolo 5

Distorsione

Come accennato nel capitolo precedente, durante l'acquisizione di una foto tramite fotocamera (Nikon) nel nostro caso, è stato utilizzato il formato RAW, per cui non sono state applicate *funzioni correttive* e dunque saranno presenti *distorsioni* che andranno corrette. I due tipi di distorsione più comuni sono:

- Distorsione radiale[20]: le linee appaiono curve, questa distorsione si accentua man mano che ci si sposta dal centro dell'immagine (un classico esempio è dato dall'effetto fish eye presente su una qualsiasi action camera)

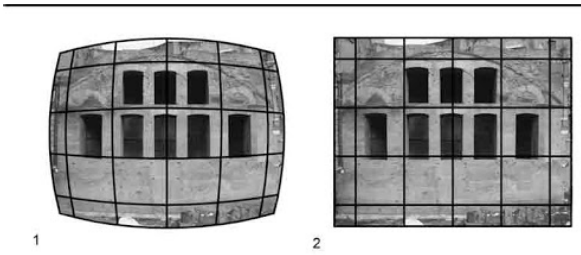

Figura 5.1: esempio di distorsione radiale [**?**]

- Distorsione tangenziale [**?**]: di solito si verifica principalmente perché le lenti non sono parallelamente allineate al piano dell'immagine, ciò fa si che l'immagine sia allungata più del dovuto oppure ruotata, ciò fa apparire gli oggetti più lontani o più vicini di quanto non siano. Di solito viene trascurata in quanto è di diversi ordini di grandezza inferiore a quella radiale.

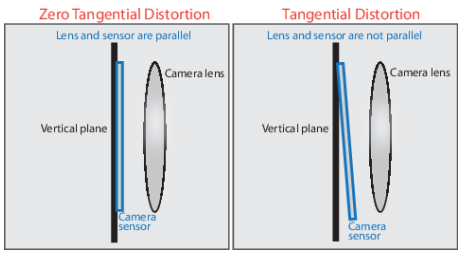

Figura 5.2: esempio di distorsione tangenziale [20]

Dunque per ridurre la distorsione bisogna fare affidamento su cinque parametri detti *Coefficienti di distorsione*, i quali riflettono la quantità di distorsione radiale e tangenziale.

```
1     Distcoeff=(k1 k2 p1 p2 k3)
2
```

$k1$, $k2$ e $k3$ rappresentano la distorsione radiale mentre $p1$ e $p2$ rappresentano la distorsione tangenziale.

Dopo aver eseguito la calibrazione è possibile procedere con l'utilizzo della funzione *undistort*[21] che provvederà a correggere l'immagine che si utilizzerà per la ricostruzione. Prima di ciò però, si può raffinare la *matrice di camera*[17] utilizzando:

```
1     h, w = img.shape[:2]
2     newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w, h), 0, (w
      , h))
3
```

la funzione *newcameramatrix* permette di ridurre al minio i pixel dell'immagine che non ci interessano, dunque potrebbero anche essere eliminati dei pixel agli angoli dell'immagine.

Per eseguire l'undistort su un'immagine OpenCV ci permette di usare due metodi:

- cv2.undistort: è il metodo più breve, chiama la funzione e utilizza il ROI ottenuto in precedenza per "croppare" il risultato

```
1     # undistort
2 dst = cv.undistort(img, mtx, dist, None, newcameramtx)
3 # crop the image
4 x, y, w, h = roi
5 dst = dst[y:y+h, x:x+w]
6 cv.imwrite('calibresult.png', dst)
7
```

- remapping: Il processo consiste nel prendere dei pixel da una porzione di immagine e di riposizionarli in una nuova posizione, in una nuova immagine. Per completare il processo di mapping, potrebbe essere necessario eseguire qualche interpolazione poiché è possibile che non ci sia sempre una correlazione dei pixel 1 ad 1. la funzione che esprime il remapping per ogni posizione dei pixel è: $g(x,y)=f(h(x,y))$ dove g è la nuova immagine, f è l'immagine sorgente ed infine $h(x,y)$ è la funzione di mappatura che opera su x,y

```
1     mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None,
      newcameramtx, (w, h), 5)
2     dst = cv2.remap(img, mapx, mapy, cv2.INTER_LINEAR)
3
```

  In questo caso è stato utilizzato il secondo metodo, tuttavia bisogna dire che non ci sono differenze importanti tra i due, per cui si è liberi di utilizzare indifferentemente entrambi i metodi. Un esempio del risultato finale è apprezzabile nella **??**

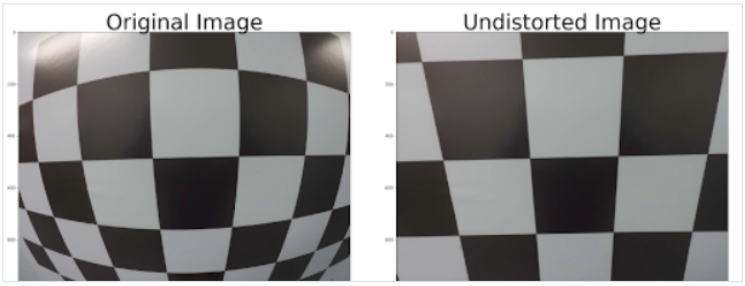

Figura 5.3: Rappresentazione della *Immagine a cui è stato applicato il metodo undistort*

Capitolo 6

Depth

La creazione di una *depth map*[22] è il passo antecedente alla generazione del *point cloud*. Ci permette di individuare la profondità degli elementi presenti all'interno della scena. Nella computazione grafica 3D, una *mappa di profondità* è un'immagine o un insieme di immagini che contiene informazioni relative alla distanza delle superfici di un oggetto da un determinato punto di vista.

6.1 Come ottenere una *depth map*

Avendo a disposizione un'unica foto, proveniente dalla camera utilizzata, non si è in grado di valutare la distanza tra la camera e gli oggetti che compaiono. Il problema non sussiste se si utilizzano due punti di vista per analizzare la scena, ovvero se si ha a disposizione una visione *stereo*[23]. Questo ci permetterà di applicare la teoria della geometria epipolare[21][24], che ci consente di descrivere le relazioni geometriche che legano due foto bidimensionali alla stessa scena tridimensionale. Supponiamo di voler fotografare una situazione come quella presente nelle immagini 6.2 e 6.1, l'oggetto in posizione **x**. Tale punto verrà proiettato sul piano immagine della fotocamera a sinistra in `x`, e a destra in `x'` nel piano immagine della fotocamera a destra, generando i punti `e` ed `e'`, che sono i rispettivi epipoli. Tale proiezione avviene tramite una trasformazione proiettiva che dipende dai parametri della camera. Se avessimo usato solamente una camera, non saremmo quindi stati in grado di ottenere tali punti. Le rispettive rette epipolari possono essere ottenute dalla proiezione della retta passante per `x - X` sul piano dell'immagine di `O` e dalla proiezione della retta passante per `x' - X` sul piano di `O'`. Dopo aver individuato due epipoli grazie all'intersezione delle rette epipolari, sarà possibile iniziare la nostra mappatura.

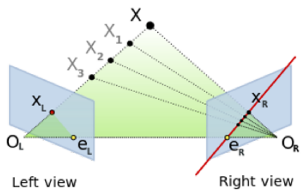

Figura 6.1: Individuazione degli epipoli[25]

La *depthmap* viene generata partendo dalla mappa della disparità e dalla posizione degli epipoli.

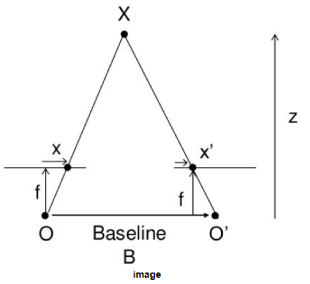

Figura 6.2: Visualizzazione ortogonale ai piani epipolari[24]

Nella figura 6.2:
`B` è la distanza tra le camere
`f` è la distanza focale (distanza oggetto-lente)
`z` è la distanza baseline-oggetto

La disparità si calcola mediante l'equazione: `x-x' = Bf/z`
Durante questa fase di acquisizione `B` ed `f` devono essere misurati con precisione: il loro valore inciderà fortemente nel calcolo della disparità. Possiamo quindi ricavare la profondità di ogni pixel all'interno dell'immagine e, confrontandola con la seconda, siamo in grado di valutare la corrispondenza degli oggetti tra le due immagini. Nei paragrafi successivi vedremo come avviene tale realizzazione con gli strumenti di OpenCV e Matlab[3].

6.2 Come calcolare una *depthmap* tramite OpenCV

Per il calcolo della *depth map* tramite la libreria OpenCV è necessario stanziare almeno un oggetto di tipo StereoSGBM nel seguente modo[26]:

```
1         left_matcher = cv2.StereoSGBM_create(minDisparity, numDisparities)
2
```

Ciò permette di implementare un algoritmo di matching semi-globale tramite pixelwise che in base ai parametri che decide di impostare risulterà più o meno preciso. I parametri indispensabili alla ricostruzione sono quelli che definiscono il *disparity range*, devono essere quindi sempre dichiarati, e solamente dopo averli definiti si può procedere con l'elaborazione. È stato deciso inoltre, di calcolare la `depth map` sfruttando tutti i parametri utili del caso, con l'intento di ottenere una qualità ottima, ed efficienza dei tempi nella ricostruzione. Se si vogliono ottenere risultati ottimi però bisognerebbe costruire un matcher e quindi una *depth map* preliminare per ogni immagine processata che sarà poi utilizzata da un filtro WLS[27] che vedremo nel capitolo 4.2.1. I matcher successivi devono essere associati ad immagini catturate alla destra di quella di riferimento. Nostro caso abbiamo quindi:

```
1         right_matcher = cv2.ximgproc.createRightMatcher(left_matcher)
2
```

La valutazione della disparità sulla singola immagine avviene tramite il metodo `stereo` che si applica nel seguente modo all'oggetto `StereoSGBM` creato in precedenza:

```
1         displ = left_matcher.compute(imageLeft, imageRight)
2         dispr = right_matcher.compute(imageRight, imageLeft)
3
```

L'output del metodo `compute` è la *mappa della disparità* preliminare che stavamo cercando:

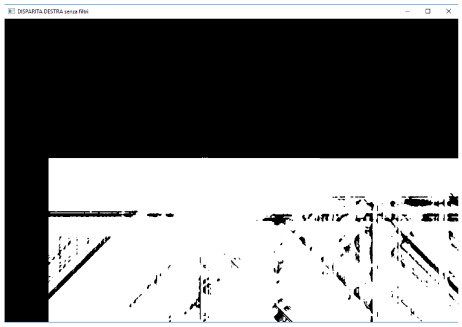

Figura 6.3: Depth senza la configurazione corretta dei parametri

6.2.1 Filtro preliminare: Weighted Least Squares filter

La CPU durante l'esecuzione di algoritmi di elaborazione di immagini stereo tende a commettere degli errori processando zone interne alle immagini con particolari caratteristiche; ad esempio nelle aree con texture omogenea, negli ambienti semi chiusi (che generano zone di ombra) e nei punti di contatto tra le variazioni di discontinuità delle profondità. È necessario quindi utilizzare almeno un filtro che ci permetta di eliminare tali errori dalla depth map. OpenCV per far fronte a questi errori, ha sviluppato il filtro WLS (weighted least squares filter), che permette tramite il confronto di depth map differenti, provenienti da una stessa scena, di assegnare dei pesi alle varie zone omogenee presenti nelle immagini, e di riconoscere e correggere gli errori sulla ricostruzione. L' effetto del filtro è il seguente:

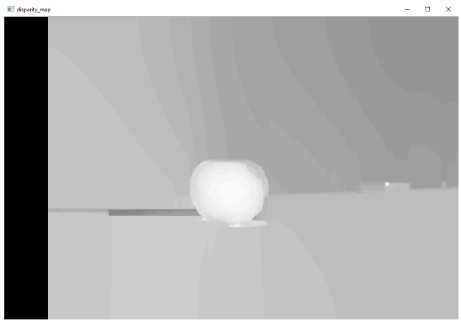

Figura 6.4: Disparità dopo l'applicazione del filtro WLS

6.2.2 Calcolo Depth map con Matlab

È stato utilizzato Matlab[3] per ottenere dei valori preliminari del range di disparità: infatti questo programma permette di calcolare il numero delle disparità e il valore minimo di disparità. Tali valori costituiscono l'inizio della configurazione dell'oggetto StereoSGBM di OpenCV. Non è indispensabile appoggiarsi a questo programma esterno, ma almeno permette di facilitare l'individuazione dei parametri ottimi. L'implementazione è la seguente:

```
1 ...
2 disparityRange = [minval maxval];
3
4 disparityMap = disparity(
5     rgb2gray(I1),
6     rgb2gray(I2),
7     'BlockSize',blockSize,
8     'DisparityRange',disparityRange
9     );
10 ...
```

Dopo aver dichiarato il DisparityRange e il BlockSize, è necessario convertire l'immagine in input in scala di grigi tramite la funzione rgb2gray() che gli deve essere passata come argomento.

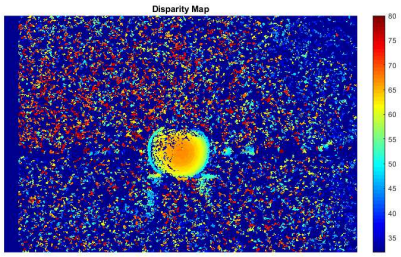

Figura 6.5: Depth map ottenuta tramite Matlab : metodo `disparity()`

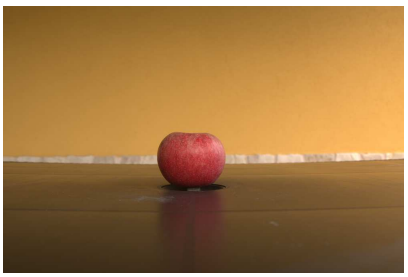

Figura 6.6: Scena ottenuta dalla camera destra

La scala di colori utilizzata in questa rappresentazione, associa colori caldi agli oggetti in prossimità della camera, e man mano che ci si allontana verranno utilizzati colori sempre più freddi. Non avendo applicato particolari filtri all'immagine processata da Matlab[3], il risultato, presenta rumore sparso.

Capitolo 7

Preparazione alla Ricostruzione

Fino ad ora sono stati utilizzati dei filtri messi a disposizione direttamente da OpenCV. Questi filtri devono essere configurati in fase di creazione della depth map perché ne influenzano fortemente la qualità. Ormai è stata prodotta una depth map accettabile e questo capitolo si concentrerà nell'individuazione e isolamento dell'oggetto di interesse dall'ambiente della scena.

7.1 Filtri usati per la ricostruzione in Python

Il software è organizzato in maniera tale da raggruppare i filtri utilizzati in due metodi indipendenti, in modo da facilitarne l'implementazione a discrezione dell'utilizzatore. Va notato che in OpenCV le immagini vengono processate come matrici di dimensione h*w*c con h altezza, w larghezza, c numero di canali. La coppia (h,w) permette di identificare la posizione di un pixel, mentre c è un vettore di dimensione variabile contenente le informazioni relative del pixel, che dipendono dalla formattazione dell'immagine (rgb,rgba, scala di grigio ...).

7.1.1 Il Goodfilter

Questo filtro è responsabile dell'individuazione degli oggetti all'interno della scena. La sua implementazione si basa sull'utilizzo di un filtro di Canny ma, prima di procedere all'applicazione si cercherà di ridurre ulteriormente il rumore presente nella scena tramite un filtro di apertura, composto a sua volta da due filtri di erosione e dilatazione, perché Canny contiene un filtro Gaussiano estremamente sensibile.

7.1.2 Filtro di dilatazione ed erosione

Questi filtri sfruttano le operazioni di trasformazione morfologica, che avviene utilizzando due immagini: quella originale e una che rappresenti (tramite una matrice 5x5 solitamente)l'intensità della trasformazione utilizzata (immagine kernel). Effettuando un confronto tra le due si ottiene l'immagine filtrata. I filtri di dilatazione e di erosione sono filtri lineari, la variazione del valore dei pixel avviene in maniera omogenea nell'area che si considera di volta in volta. Per ottenere buoni risultati è necessario applicare in ordine il filtro di dilatazione e

poi quello di erosione. Un filtro morfologico di dilatazione permette di ispessire i contorni degli oggetti presenti nell'immagine aggiungendo pixel nei bordi che li individuano. Tale operazione ha lo scopo di renderli più visibili, ma soprattutto permette di riempire i piccoli "buchi" presenti nella scena dovuti al rumore. La figura 7.1 riporta i risultati ottenuti dall'applicazione di tale filtro all'immagine in scala di grigio (ad ogni pixel viene associato un unico valore che rappresenta l'intensità della tonalità). Viene considerato un gruppo di pixel dell'immagine iniziale, ad ognuno di essi è associato un valore di tonalità di grigio. Nella costruzione dell'immagine risultante, a tutti quei pixel corrisponderà il valore più alto di tonalità ricavato da quelli considerati inizialmente.

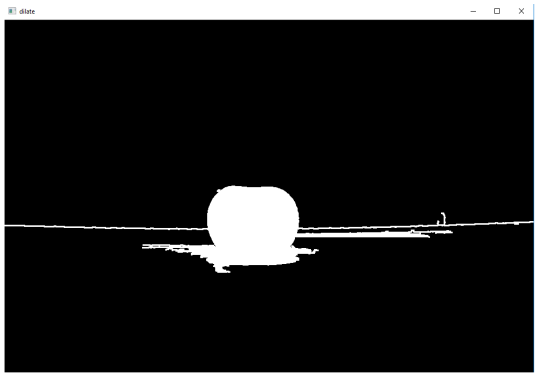

Figura 7.1: Effetto dovuto ad un applicazione iterativa del filtro di dilatazione

Un filtro morfologico di erosione diminuisce lo spessore dei bordi degli oggetti presenti nella scena operando in maniera opposta al filtro di dilatazione: il valore associato ad ogni pixel in output è ottenuto dal valore minimo posseduto dai propri vicini presenti nell'immagine in input. L'obbiettivo di questo filtro è di eliminare gli oggetti di piccole dimensioni in modo da isolare solamente l'oggetto della scena (esempio in figura)

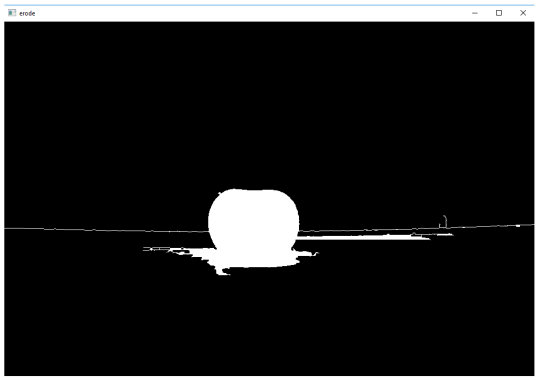

Figura 7.2: Effetto dovuto a un applicazione iterativa del filtro di erosione

Per ottenere risultati ottimali, questi due filtri vanno utilizzati insieme più volte, in modo da ottenere un filtro morfologico di apertura. In questa implementazione si è deciso di iterarne l'utilizzo con lo scopo di individuare il bordo

chiuso più grande presente nella scena per agevolare l'individuazione del soggetto dell'immagine

7.1.3 Filtro di Canny per il riconoscimento dei bordi

L'algoritmo di Canny è un operatore che permette l'individuazione dei bordi degli oggetti interni alla scena. Impostando pochi parametri, è possibile ottenere ottimi risultati in quasi tutte le situazioni. L'algoritmo si sviluppa concatenando diverse fasi, ognuna delle quali influenzerà la successiva. Inoltre, ad ogni fase dovranno essere stabiliti dei parametri di configurazione ottimi, in modo da permettere all'algoritmo l'individuazione e la selezione più accurata ed efficiente possibile. Nella scelta di questi parametri è bene confrontare sempre i bordi reali degli oggetti interessati e quelli ottenuti dall'algoritmo nelle varie configurazioni.

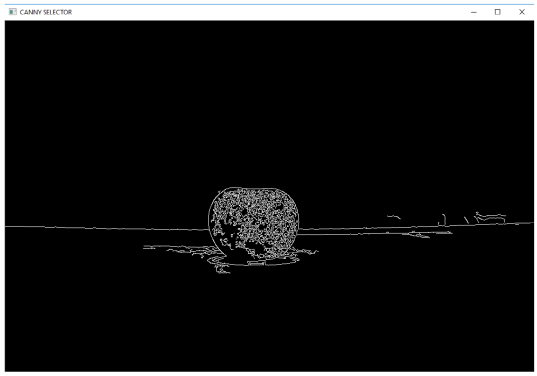

Figura 7.3: Risultato dell'analisi dei bordi tramite l'algoritmo di Canny

7.1.4 Circular filter

In alcuni casi non è bastato utilizzare il goodFilter: a causa della scarsa qualità delle foto acquisite, le zone dell'immagine dove sono presenti riflessi o particolari fasci luminosi direzionali dopo essere processate non permettono una giusta identificazione dei bordi utili alla selezione dell'oggetto. Si è deciso di utilizzare un filtro aggiuntivo che può essere implementato separatamente oppure in combinazione con il precedente. L'effetto di questo filtro è una selezione ad area circolare attorno all'immagine di interesse, ciò è possibile tramite traslazione e ridimensionamento di un cerchio visualizzato a schermo.

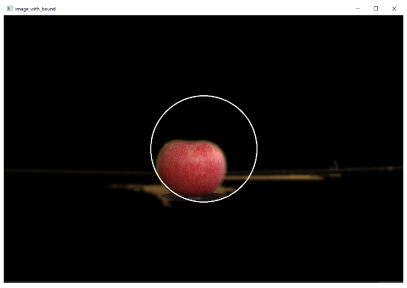

Figura 7.4: Uso del circularFilter

Figura 7.5: Situazione prima del circularFilter

Figura 7.6: Situazione dopo il circularFilter

Capitolo 8

Ricostruzione

A questo punto si dovrebbe disporre di una *depth map* di buona qualità, con all'interno della scena una situazione simile alla figura 7.6: dove è presente solamente l'oggetto in questione e poco altro. I filtri del capitolo cinque non sono indispensabili alla ricostruzione, visualizzazione e salvataggio del modello tridimensionale, ma è fortemente consigliato farne uso, poiché ogni pixel appartenente ad un'area della *depth map* influenzata da luce sfavorevole, effetti di sfocatura (blurred) oppure rumore in generale, assumerà un valore di intensità discostato dal valore effettivo, generando quindi deformità dal modello reale.

8.1 Generazione modello in Python

La creazione del *point cloud* avviene tramite *depth map*; questa però non è sufficiente ad effettuare la ricostruzione accurata dell'ambiente. Con lo scopo di migliorare la qualità della *nuvola di punti*, è stata introdotta la matrice di camera[28]. Tramite essa, si può ottenere una stima più accurata delle posizioni dei punti nell'ambiente 3D. Grazie alle misure effettuate in fase di acquisizione, possiamo costruire tale matrice. I parametri necessari alla costruzione della matrice di camera sono:

- la distanza dell'oggetto dalla lente della camera
- la distanza tra le due camere
- le dimensioni delle immagini utilizzate

```
1     q =     [1, 0, 0, -width / 2],
2             [0, 1, 0, -height / 2],
3             [0, 0, 0, focal_lenght],
4             [0, 0, -1 / distanza_camere, 0]
```

Il suo utilizzo è favorito dal metodo specificatamente sviluppato da OpenCV che permette di ottenere la posizione dei punti nell'ambiente 3D senza dover ridimensionare manualmente l'immagine[28] :

```
1     points = cv2.reprojectImageTo3D(disp, q)
2
```

I colori associati ai punti che la compongono possono essere facilmente ricavati da un confronto con l'immagine in input. Per valutare temporaneamente la qualità della ricostruzione e quindi la correttezza dei parametri della matrice di camera prima di salvare ed esportare il modello, è consigliabile affidarsi a un metodo di visualizzazione in ambiente di sviluppo locale sviluppato da OpenCV, che partendo dalle posizioni dei punti dello spazio tridimensionale è in grado di ricavare le coordinate rapportate ad un ambiente bidimensionale, in modo da ricostruire una visione prospettica della costruzione 3D. I parametri necessari all'elaborazione sono i punti nel piano 3D `3Dpoints`, una matrice necessaria alla rotazione prospettica `r`, una alla traslazione `t`, una matrice di correzione degli errori generati dalla camera `k` e i coefficienti di distorsione associati alla camera `distcoeff`.

```
1     ...
2     projected, temp = cv2.projectPoints(3Dpoints, r, t, k, distcoeff)
3     ...
4
```

Immettendo tale metodo in un ciclo seguito da una visualizzazione istantanea si è in grado di navigare all'interno della ricostruzione generando ad ogni variazione di r,t una trasformazione che permette all'utente di osservare la qualità del modello ottenuto. L'effetto di tale feature è riscontrabile nelle figure 8.1,8.2, 8.3.

```
1         view3dPointCloud(points, colors, k, dist_coeff, width, height)
2
```

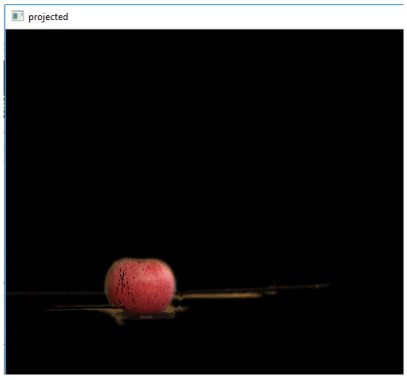

Figura 8.1: Visualizzazione in ambiente OpenCV: view1

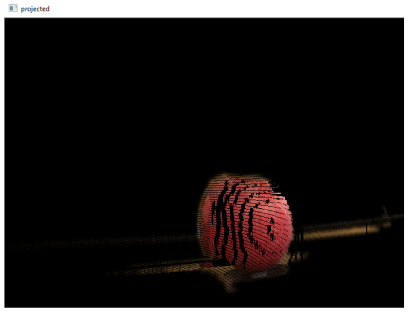

Figura 8.2: Visualizzazione in ambiente OpenCV: view2

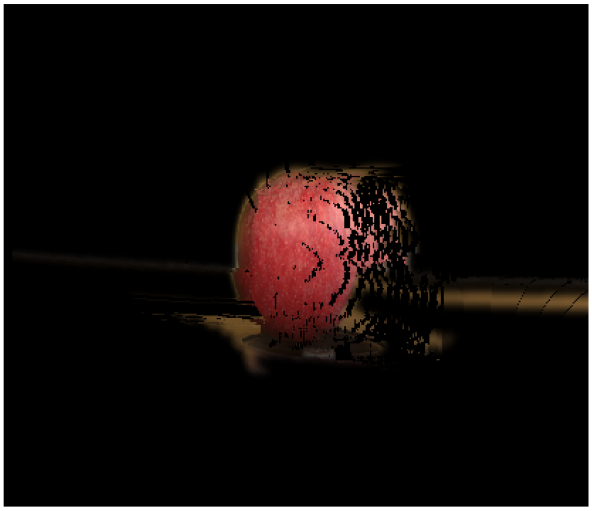

Figura 8.3: Visualizzazione in ambiente OpenCV: view3

8.1.1 Salvataggio del modello

Ottenuta una ricostruzione soddisfacente del modello, lo si può salvare in qualunque formato si voglia grazie alla facile reperibilità delle informazioni riguardo ai pixel (posizione nello spazio 3D e colore associato). In questo elaborato il salvataggio può avvenire sia in formato .Las[10] che in .Ply[7], come già spiegato nel capitolo due. Per esportare il modello in tali formati, è necessario effettuare un lavoro preliminare in modo da creare correttamente il file.

8.1.1.1 Generazione del file.ply

Per la creazione del .ply abbiamo strutturato un metodo che adatta il formato dei punti e dei colori prodotto in fase della elaborazione precedente in quello del .ply standard

```
1         #questo    il metodo di scrittura standard del .ply, che avviene
      tramite l'utilizzo dell'header.
2         def write_ply(nome_file, vertici, colori):
3             ...
```

```
4             with open(nome_file, 'wb') as f:
5                 f.write((ply_header % dict(vert_num=len(verts))).encode('utf-8
      '))
6                 print((ply_header % dict(vert_num=len(verts))).encode('utf-8')
      )
7                 np.savetxt(f, vertici, fmt='%f %f %f %d %d %d')
8
9             ...
10
```

Il formato dell'header del .ply è strutturato in questo modo:

```
1         ply_header = '''ply
2         format ascii 1.0
3         element vertex %(vert_num)d
4         property float x
5         property float y
6         property float z
7         property uchar red
8         property uchar green
9         property uchar blue
10        end_header
11
12        '''
13
```

8.1.1.2 Generazione del file Las

Il file .Las ha una struttura più complicata[11][29]. La lettura e la manipolazione delle informazioni non è gestita da un unico formato di header inoltre i dati risultano compressi poiché è uno standard per file di grandi dimensioni. Proprio per ottimizzare il salvataggio di oggetti estremamente grandi abbiamo deciso di sequenzializzare la scrittura dei vertici con l'appoggio di una classe specifica

```
1         ...
2             class newPoint():
3         ...
4          #tramite un buffer per l'inserimento
5
6         ...
7             class pointBuf():
8         ...
9
```

Per non appesantire l'esecuzione del metodo. Tale scelta dà possibilità all'utilizzatore di cambiare senza troppe difficoltà la versione del formato dell'header del .Las per aggiungere eventualmente altre informazioni (come la data di cattura). In fase di creazione del .Las è importante specificare la versione dell'header: sono presenti moltissime strutture che permettono di implementare differenti features. In questo caso si è stabilito che fosse conveniente utilizzare la versione 1.2. La versione deve essere specificata in questo modo.

```
1         ...
2         header.format = 1.2
3         header.data_format_id = 2
4         ...
5
```

L'effetto di tali parametri genera la struttura 8.4.

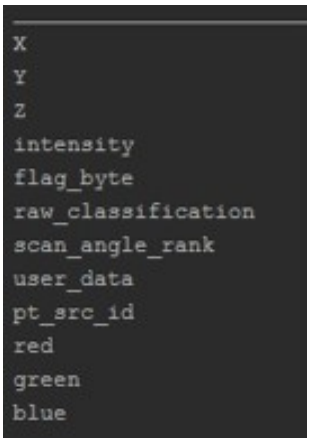

Figura 8.4: Struttura del .las con header V1.2 e struttura dei dati V2

Analogamente, se si presentano problemi di visualizzazione, oppure se si hanno esigenze particolari, sarebbe corretto specificare la dimensione minima dei gruppi di vertici da considerare lungo le varie direzioni (X,Y,Z).

```
1         ...
2             lasfile.header.offset = [xmin, ymin, zmin]
3         ...
4
```

8.1.2 Visualizzazione in Potree

La visualizzazione del modello 3D del *point cloud*, avviene in server locale Apache, gestito tramite Xampp[4]. Una volta avviato e configurato Xampp è possibile utilizzare il web server. Ora non rimane che caricare il modello 3D nel server e visualizzarlo tramite Potree[1]. La fase di visualizzazione però, è preceduta da una conversione del modello. Per la visualizzazione tramite Potree, è necessario convertire il file contenente il *point cloud*, da .las o .ply in una struttura *octree* (albero in cui ogni nodo ha esattamente otto figli) tramite l'eseguibile *Potree-Converter*. In questa implementazione, dopo una prima configurazione, è stata automatizzata la visualizzazione del modello. È sufficiente specificare i percorsi degli eseguibili, di salvataggio dei file e il nome della pagina generata.

Nelle figure 8.5 e 8.6 si possono notare rispettivamente le interfacce di configurazione di *Xammpp* e *Potree*

```
1     ...
2         os.system(potree_converter_path+' '
3         +potree_saving_las_path+'
4         --overwrite -o '
5         +xampp_folder+'
6         -d 400 --output-format LAS -q NICE --edl-enabled --show-skybox --
      generate-page '
7         +name_of_generated_page)
8     ...
9
```

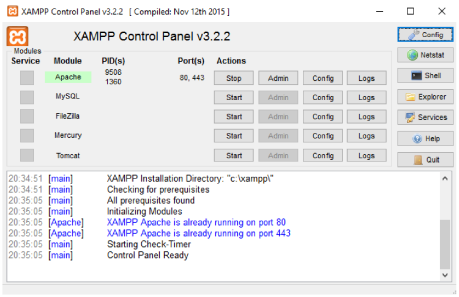

Figura 8.5: interfaccia di configurazione Xampp

Figura 8.6: Interfaccia di Potree

Figura 8.7: Potree: particolare1

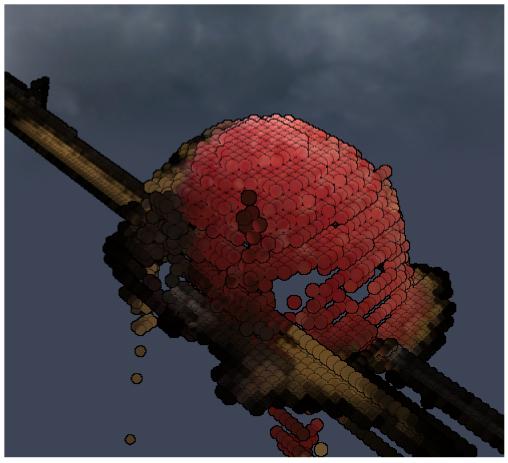

Figura 8.8: Potree: particolare2

8.2 Generazione modello Metashape

Il software professionale Metashape[**?**] della Agisoft, permette di ricostruire il modello tridimensionale, partendo da un dataset di foto, di dimensione variabile. Il processo di ricostruzione, ovvero il *workflow*, è indipendente dal numero di foto utilizzate. Ovviamente, più foto si hanno a disposizione, migliore sarà la ricostruzione, a discapito dei tempi di esecuzione. Il *workflow* è strutturato nel seguente modo[30]:

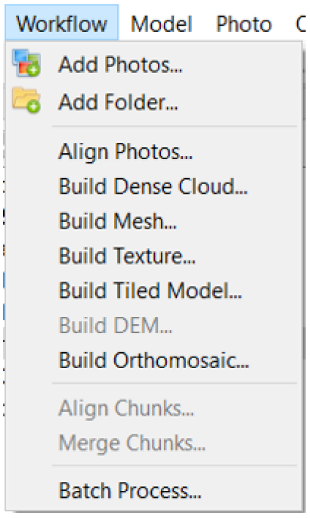

Figura 8.9: Workflow di Metashape: rendering basato su *point cloud*

- la prima fase è ovviamente quella di selezione dei file da dare in input al programma. Non ci sono condizioni particolari che vincolano il formato delle immagini; è preferibile ma non indispensabile caricarle in .raw, poiché è un formato lossless, e affidare l'applicazione della correzione direttamente al programma, che in base allo strumento di cattura utilizzato, caricherà direttamente i parametri ottimali per la correzione.

- il passo successivo è l'allineamento delle foto. Il programma è in grado di individuare le rette epipolari presenti nelle varie coppie di immagini, in modo da allinearle in uno spazio tridimensionale. I loro punti di intersezione individueranno i vari epipoli, dai quali sarà possibile risalire alle posizioni dei punti di acquisizione delle foto. Il risultato di questa fase è la visualizzazione a schermo delle zone di cattura delle foto. A volte può succedere, sopratutto quando si ha a che fare con oggetti sferici e un background ridotto, che la posizione dei punti di cattura venga ruotata di 180° rispetto al soggetto della scena: in tal caso le foto associate a tali punti devono essere scartate manualmente. Non tutte le immagini sono sempre accettabili, poiché il software non è sempre in grado di ricavare le rette di epipolo. Solitamente questo succede se l'immagine non è stata acquisita correttamente (effetti di sfocatura, fasci luminosi, rumore eccessivo ...).

- ora si inizia la realizzazione del modello 3D. Lanciando il comando di costruzione della nuvola di punti il programma assegna la profondità ai pixel della scena effettuando confronti tra le depth ottenute nelle varie coppie di foto. Il risultato di ciò è il modello tridimensionale dei punti della scena ovvero la nuvola di punti (*point cloud*).

- il SW per migliorare la ricostruzione, prima di applicare la texture calcola una mesh, cioè la superficie che contiene tale nuvola di punti. Dopo tale valutazione potrà avvenire il rendering 3D:

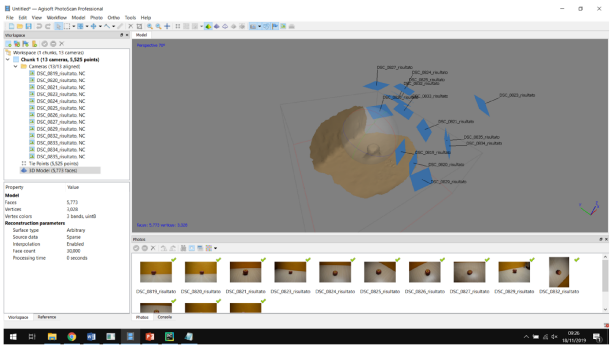

Figura 8.10: Metashape: fase di mashing

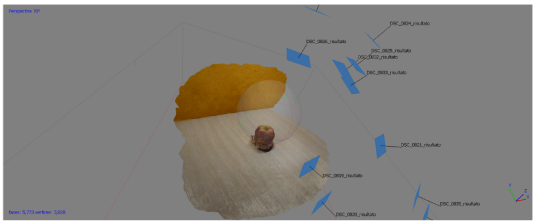

Figura 8.11: 3dmodel- final reconstruction

Ricavato il modello tridimensionale, lo si può manipolare manualmente eliminando regioni di punti non utili dalla scena oppure automaticamente. La manipolazione automatica va fatta specificando vari parametri di intensità e precisione durante le fasi indicate.

8.3 Generazione del modello tramite Kinect

Il kinect[15] è un hardware prodotto dalla Microsoft specializzato nella *computer vision*. La ricostruzione 3D di un oggetto avviene in maniera del tutto automatica tramite l'apposito software. È sufficiente che non ci siano oggetti in movimento nella scena. Il dispositivo durante la fase di cattura delle immagini deve essere spostato nell'area o ruotato progressivamente attorno alla zona dell'esperimento: così facendo il Kinect catturerà in maniera del tutto automatica una sequenza di fotogrammi, i quali verranno poi utilizzati nella ricostruzione dell'ambiente basandosi sul matching dei pixel appartenenti ad aree di simili dimensioni. I risultati sono le figure 8.12,8.13

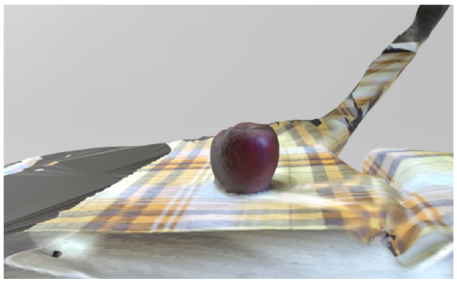

Figura 8.12: ricostruzione tramite Kinect: punto di vista1

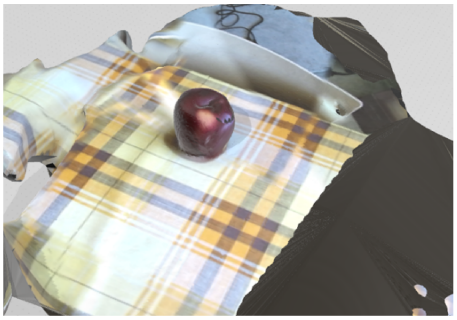

Figura 8.13: ricostruzione tramite Kinect: punto di vista2

Capitolo 9

Conclusione e sviluppi futuri

9.1 Conclusione

La ricostruzione tridimensionale di frutti partendo da una configurazione di immagini stereo, figura 3.2 e 6.6, è avvenuta con discreti risultati. Questo progetto è stato diviso in step, dunque è molto importante rispettare le varie fasi. Già dalla fase di acquisizione delle immagini, ci si è resi conto della necessità di effettuare una particolare verifica preliminare dell'ambiente di cattura. Fin dai primi test effettuati su mele, arance e tazze si è notata la vulnerabilità di questo procedimento alle sfocature, ai raggi direzionali di luce intensa e soprattutto ai riflessi generati dalla superficie dell'oggetto della foto. Il problema è stato quasi del tutto risolto prendendo precauzioni preliminari, come l'utilizzo di luce diffusa e posizionamento dell'oggetto di cattura in ambienti assorbenti, e provvedimenti in corso d'opera, come l'aggiunta di filtri di vario tipo.

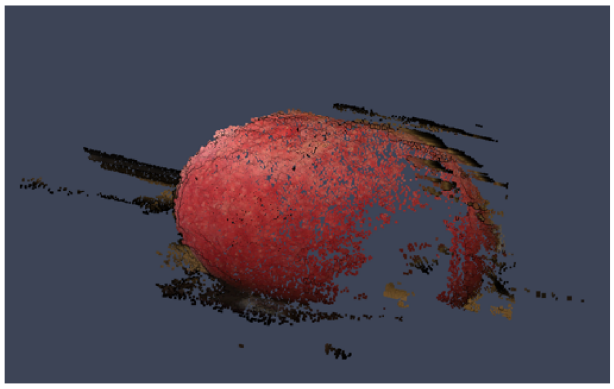

Figura 9.1: Effetto di un fascio luminoso nella ricostruzione

Nella fase di processamento delle immagini, cioè calcolo della *depth map* e ricostruzione, a causa della limitatezza delle risorse hardware è stato necessario effettuare una conversione in .png (loseless), effettuata tramite il software ImageMagic[31], e ridimensionare le immagini tramite apposito script. Così facendo si è passati dall'utilizzare coppie di foto di dimensione 4000x6000 in .NEF (formato grezzo della fotocamera utilizzata) ad un 600 x 1000 in .png. Ciò ha sicuramente inficiato nella qualità della ricostruzione finale.

9.2 Confronto tra i modelli ottenuti

I risultati ottenuti dai diversi procedimenti, mettono in luce i limiti della realizzazione del modello 3D tramite OpenCV. Tale modello presenta infatti degli errori di ricostruzione: come la mancanza di alcuni punti all'interno del frutto e la presenza di altri appartenenti al background 9.1. Bisogna però non trascurare diverse considerazioni:

- gli strumenti provenienti da OpenCV sono open source e gratuiti, inoltre in questo esperimento, è stata utilizzata una sola coppia di foto ad ogni ricostruzione. Nella corretta esecuzione della ricostruzione, è necessario effettuare le misure raccomandate nel capitolo 3, che devono essere eseguite in un ambiente idoneo. Osservando l'immagine delle figure 6.6 e 9.1, si notano direttamente gli effetti di un ambiente dannoso alla costruzione.

- il Kinect invece ha un software di licenza closed source; è certificato Microsoft la quale mette a disposizione ambienti di configurazione specifici per tale piattaforma, garantendo qualità nella ricostruzione ed efficienza nei tempi di acquisizione ed elaborazione.

- il software professionale Metashape risulta con licenza closed source. Differentemente dal Kinect permette di intervenire in fase di elaborazione, e quindi dà la possibilità all'utente di scegliere tra parametri qualitativi per la ricostruzione, che risulta automatizzata ed estremamente corretta: infatti abbiamo ottenuto ottimi modelli anche processando solamente due foto. Questo software è molto interessante perché oltre alla costruzione del modello permette, se necessario, di approssimare la costruzione di zone non visibili dalle camere con ottimi risultati.

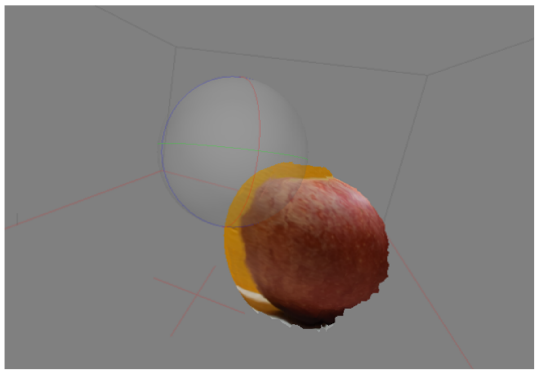

Figura 9.2: Photoscan: risultato della ricostruzione utilizzando 15 foto

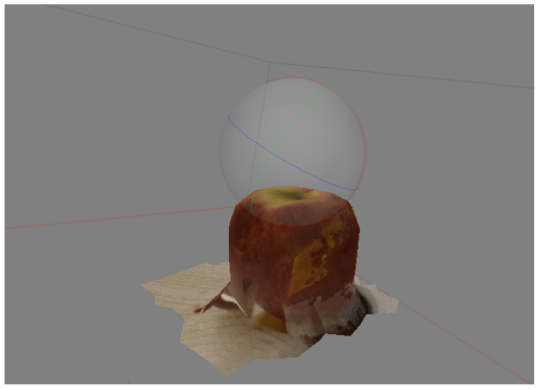

Figura 9.3: Photoscan: risultato della ricostruzione utilizzando 15 foto

9.3 Sviluppi Futuri

Come si può intuire leggendo questa tesi, ottenere un modello 3D perfetto e senza difetti è pressoché impossibile o comunque molto difficile. Tuttavia c'è un modo efficace per ottenere un risultato migliore, a patto che vengano rispettati i parametri noti durante l'acquisizione dell'immagine (discussi nel capitolo 3). Il metodo in questione è l'utilizzo di una *depth camera*, come ad esempio la *Intel RealSense Camera D455*[32]. Questa Camera, la quale utilizza tecnologia Stereoscopica, permette di ottenere una *Depth Resolution* fino a 1280x720 ed una *depth accuracy* altissima con un errore di massimo il 2 percento fino a 4 metri di distanza. Il *Range* ideale va da 60 cm a 6 metri. La qualità dell'acquisizione è ottima soprattutto grazie al proprio Hardware, monta un *Intel RealSense depth module D450*[32].

Figura 9.4: Intel RealSense Camera D455

Bibliografia

[1] Markus Schütz. *Potree: Rendering Large Point Clouds in Web Browsers*. PhD thesis, 09 2016.

[2] Opencv team. Opencv. https://opencv.org, 2019.

[3] MATLAB. *version R2019b*. The MathWorks Inc., Natick, Massachusetts, 2018.

[4] Apache. Xampp apache. https://www.apachefriends.org/it/index.html.

[5] Cory Althoff Michael Kennedy. Jetbrains, pycharm. https://www.jetbrains.com/pycharm/.

[6] Travis Oliphant. *Guide to NumPy*. 01 2006.

[7] Greg Turk. The ply polygon file format. https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html, 2012.

[8] American Society for Photogrammetry and Remote Sensing (ASPRS). Las specification version 1.4. https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf, 2011.

[9] Progetto pst - dati lidar. http://www.pcn.minambiente.it/mattm/progetto-pst-dati-lidar.

[10] ASPRS. Las specification version 1.2. https://www.asprs.org/wp-content/uploads/2010/12/asprs_las_format_v12.pdf, 2008.

[11] Grant Brown. Las implementation. https://github.com/grantbrown/laspy, 2012-2019.

[12] Atlassian. Bitbucket. https://bitbucket.org/, 2008.

[13] Humboldt State University. Structure from motion. http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson8-2/SfM.html, 2014.

[14] Agisoft metashape user manual professional edition.

[15] Microsoft. Kinect. https://developer.microsoft.com/it-it/windows/kinect, 2014.

[16] MathWorks. Camera calibration. https://it.mathworks.com/help/vision/ug/camera-calibration.html, 2018.

[17] Opencv Dev team. Camera calibration. https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html, 2014.

[18] Wikipedia Contributors. Computer vision. https://en.wikipedia.org/wiki/Computer_vision, 2021.

[19] Wikipedia Contributors. Stereoscopia. https://it.wikipedia.org/wiki/Stereoscopia, 2014.

[20] Wikipedia Contributors. Distorsion (optics). ://en.wikipedia.org/wiki/Distortion_(*optics*), 2014.

[21] opencv dev team. epipolar geometry. https://docs.opencv.org/master/da/de9/tutorial_py_epipolar_geometry.html, 2013.

[22] Heiko Hirschm¨uller. Stereo processing by semi-global matching and mutual information. https://core.ac.uk/download/pdf/11134866.pdf, 2007.

[23] Stefano Mattoccia. Stereo vision: Algorithms and applications. http://vision.deis.unibo.it/~smatt/Seminars/StereoVision.pdf, 2013.

[24] opencv dev team. Opencv depth construction. https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html, 2014.

[25] Arne Nordmann. Epipolar geometry. https://commons.wikimedia.org/wiki/File:Epipolar_geometry.svg, 1 gennaio 2007.

[26] opencv dev team. Stereosgbm class reference. https://docs.opencv.org/master/d2/d85/classcv_1_1StereoSGBM.html, 2014.

[27] opencv dev team. Disparity map post-filtering. https://docs.opencv.org/master/d3/d14/tutorial_ximgproc_disparity_filtering.html, 2013.

[28] opencv dev team. reprojection structure. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, 2014.

[29] Grant Brown. Las implementation. https://laspy.readthedocs.io/en/latest/tut_background.html, 2012-2019.

[30] Agisoft LLC. Agisoft photoscan manuale d'uso. https://www.agisoft.com/pdf/manuals_other/pscan_pro_it_1-2.pdf, 2016.

[31] Wikipedia Contributors. Imagemagik. ://en.wikipedia.org/wiki/ImageMagick, 2014.

[32] Intel. D455. ://www.intelrealsense.com/depth-camera-d455/, 2020.

[33] opencv dev team. opencv-python-tutorials. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html, 2013.

[34] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–.

[35] MathWorks. Disparity with matlab. https://it.mathworks.com/help/vision/ref/disparity.html, 2018.

[36] Alexander Mordvintsev. Canny edge detection. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html, 2018.

[37] Justin Liang. Canny edge detection. http://justin-liang.com/tutorials/canny, 1994-2016.

[38] opencv dev team. Morphological transformations. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html, 2018.

[39] Samuel Rota Bulò. Corso di visione artificiale. https://www.dsi.unive.it/~srotabul/files/vision/Visione-05-Filtri-parte1.pdf, 2010.

[40] MathWorks. disparity. https://it.mathworks.com/help/vision/ref/disparity.html, 1994-2019.

[41] MathWorks. Types of morphological operations. https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html, 1994-2019.

[42] opencv dev team. Canny edge detection. https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html, 2014.

[43] Wikipedia contributors. Sum of absolute differences. https://en.wikipedia.org/w/index.php?title=Sum_of_absolute_differences&oldid=683158983, 2019.

[44] Python documentation.

[45] Margaret Rouse. Point cloud library. https://whatis.techtarget.com/definition/point-cloud, 2018.

[46] opencv dev team. Disparity filter original paper. https://github.com/opencv/opencv_contrib/blob/2de6ff576c75ab96e10f5b38dfcfeae013471b63/modules/ximgproc/include/opencv2/ximgproc/disparity_filter.hpp#L78, 2015.

[47] opencv dev team. Camera calibration and 3d reconstruction. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html, 2014.

Elenco delle figure