UNIVERSITÀ
degli STUDI
di CATANIA

Calibrazione di due camere ed un sistema stereoscopico

Fabio Merola W82000188. Professore: Sebastiano Battiato.

Dipartimento: D.M.I. Informatica LM-18 A.A. 2019/2020

Data documento: 11 Aprile 2020.

Indice

1 Introduzione

Lo scopo di questo elaborato è quello di riuscire a calibrare due camere ed un sistema stereoscopico avvalendosi della libreria OpenCV tramite script in Python.Con una delle camere calibrate, infine, tentare di eseguire una misurazione delle dimensioni di un oggetto reale. Trarne quindi delle conclusioni che prendano in considerazione ambedue gli aspetti fondamentali: le camere e la suddetta libreria. Il progetto è quindi diviso in tre fasi principali:

- Raccolta dati;
- Sviluppo script;
- Interpretazione dei risultati.

2 Raccolta dati

Parte fondamentale del progetto, la raccolta dati è stata attuata con le camere di due smartphone, sorretti da strutture fisse sia in rotazione che traslazione. La posizione di quest'ultime, come sarà possibile vedere nei risultati, è stata scelta in modo da poter usare lo stesso set fotografico sia per la calibrazione delle singole camere, che per la calibrazione del sistema stereoscopico da esse composto. Nel dettaglio le camere compongono un sistema stereoscopico ad assi convergenti. Il soggetto del set è una scacchiera, elemento che, grazie alle sue caratteristiche, facilita le calibrazioni tramite OpenCV.

2.1 Set fotografico

Il set è composto da 42 fotografie, 21 per camera, della scacchiera in differenti posizioni. L'esperimento è stato svolto su più copie di questo set, elaborato al fine di comprendere il peso che OpenCV assegna ad alcune caratteristiche principali dell'immagine. Nel dettaglio le variabili delle immagini prese in considerazione sono:

- Dimensione(100%, 75%, 50%, 25%, 10%);
- Canali colore(RGB, Grayscale);
- Contrasto(+15%, +30%).

2.2 Dispositivi utilizzati

Le due camere utilizzate, destra e sinistra, appartengono rispettivamente ad un Samsung Galaxy S8(SM-G950F) ed un Huawei Mate 20 Pro(LYA-L29). Queste le specifiche:

Caratteristiche	**SM-G950F**	**LYA-L29**
Formato immagine	jpg(JPEG)	jpg(JPEG)
Risoluzione [4:3]	12 MP	19 MP
Valore d'apertura	1,53 EV (f/1,7)	2,27 EV (f/2,2)
Modalità esposimetro	Media pesata al centro	Modello
Lunghezza focale	4,2 mm	2,4 mm
Software	G950FXXS8DTC1	LYA-L29 10.0.0.189(C432E7R1P5)

3 Sviluppo script

Tutti gli script sono sviluppati in Python 2.7 per necessità ma assolutamente utilizzabili in Python 3.x. Questi sono disponibili, insieme alle analisi principali su GitHub[1]. Le librerie utilizzate sono alle seguenti versioni:

Libreria	**Versione**
OpenCV	4.2.0
Matplotlib	2.2.5
Numpy	1.15.2
PIL	5.3.0

3.1 Pre-processing

Il set fotografico è stato elaborato tramite la libreria PIL in Python al fine di ottenere le copie previste.

```
from PIL import Image, ImageEnhance
img = Image.open(image_path)
```

Per il ridimensionamento delle immagini la funzione utilizzata è:

```
size_factor = 0.75
width, height = img.size #Dimensioni originali
newsize = (int(width*size_factor), int(height*size_factor))
img = img.resize(newsize)
img.save(image_path)
```

Per la conversione da RGB a Grayscale:

```
img = img.convert('L')
img.save(image_path)
```

Mentre per l'aumento del contrasto:

```
enhancer = ImageEnhance.Contrast(img)
enhancer.enhance(1.30).save(image_path) # 1.30 = +30%
```

3.2 Calibrazione singola camera

Per lo script risulta necessario importare OpenCV e Numpy:

```
import numpy, cv2
```

I punti reali della scacchiera, considerata una matrice di 7x7 (righe x colonne) e la dimensione del lato di ogni quadrato di 2 centimetri, sono stati generati nel seguente modo:

```
real_points = numpy.array([
        [numpy.float32(j),numpy.float32(i),numpy.float32(0)]
        for i in range(0, 14, 2) for j in range(0, 14, 2)
])
```

Per ogni immagine del set preso in esame si esegue la funzione di OpenCV in grado di trovare la scacchiera nell'immagine:

```
img = cv2.imread(IMG_PATH)
ret, corners = cv2.findChessboardCorners(img, (7,7))
```

Infine si calcola la calibrazione della camera utilizzando nuovamente una funzone di OpenCV, dandogli in input le dimensioni di un'immagine(comuni a tutto il set), l'insieme dei punti `p2d` trovati nelle immagini e in egual numero l'insieme dei punti reali `p3d` calcolati precedentemente:

```
ret, mtx, dist, rvecs, tvecs = (
        cv2.calibrateCamera(p3d, p2d, (height,width),None, None)
)
```

3.3 Calibrazione sistema stereoscopico

In questo caso, per motivi computazionali, si è deciso di utilizzare un solo set di immagini (camera destra e camera sinistra), in particolare si è optato per quello che ha dato migliori risultati nella fase precedente. Il fulcro dello script, ovvero le righe che sostanzialmente variano rispetto allo script precedente sono:

```
retval, _, _, _, _, R, T, E, F=(
        cv2.stereoCalibrate(
                p3d,
                p2d_left,
                p2d_right,
                mtx_l,
                dist_l,
                mtx_r,
                dist_r,
                (max(height_l,height_r),max(width_l,width_r)),
                flags=cv2.CALIB_FIX_INTRINSIC
        )
)
```

- Funzione che si occupa della calibrazione del sistema stereoscopico, prendendo in input le calibrazioni delle singole camere ed i vari insiemi di punti visti precedentemente;

```
R1, R2, P1, P2, Q, roi1, roi2 =(
        cv2.stereoRectify(
                mtx_l,
                dist_l,
                mtx_r,
                dist_r,
                (max(height_l,height_r),max(width_l,width_r)),
                R,
                T
        )
)
map1_x,map1_y=(
        cv2.initUndistortRectifyMap(
                mtx_l,
                dist_l,
                R1,
                P1,
                (max(height_l,height_r),max(width_l,width_r)),
                cv2.CV_32FC1
        )
)
img_left_remapped=cv2.remap(
        img_left,
        map1_x,
        map1_y,
        cv2.INTER_CUBIC
)
x_l,y_l,width_l,height_l = roi1
img_left_remapped =(
    img_left_remapped[
        :max(width_l,width_r),
        :max(height_l,height_r)
    ]
)
```

- Questo invece per quanto riguarda la rettifica delle immagini. Si noti che le funzioni prendono come input i risultati delle funzioni precedenti, terminando in un "crop" dell'immagine che escluda contenuto inutile ed uniformi le dimensioni delle immagini acquisite con differenti risoluzioni.

Nel dettaglio, controllando la documentazione e comparando i risultati si è trovato più opportuno usare gli output della:

`cv2.stereoRectify(...)`

come input della funzione:

`cv2.initUndistortRectifyMap(...)`,

piuttosto che gli output della funzione:

`cv2.getOptimalNewCameraMatrix(...)`

consigliata a lezione.

Inoltre, in questo caso, per ottenere la calibrazione della singola camera è stato necessario, come dimostrato nelle conclusioni, utilizzare una funzione equipollente alla:

`cv2.findChessboardCorners(...)`

che ritornasse però risultati con precisione superiore:

`cv2.findChessboardCornersSB(...)`[2]

3.4 Misurazione oggetto reale

Sfruttando le conoscenze apprese a lezione sulle equazioni riguardanti il *Pinhole camera model*, si è optato per un utilizzo delle loro inverse per trasformare dei punti nel dominio dell'immagine(pixel) in punti del mondo reale(cm). Quindi si è proceduto a misurare delle distanze note, quelle tra due punti della scacchiera, in modo da poterle trattare in una comparativa.

Nel dettaglio si è proceduto prima alla stima di un fattore di scala medio per il piano della scacchiera:

```
#Rotation matrix from rotation vector
rmatrix, jacobian = cv2.Rodrigues(rvecs[3])
#Rotation matrix added of traslation vector
Rt = numpy.column_stack((rmatrix,tvecs[3]))
P_mtx = mtx.dot(Rt) #Projective matrix
s_sum = 0
for i in range(len(real_points)):
        XYZ1=numpy.array([[
             real_points[i,0],
             real_points[i,1],
             real_points[i,2],
             1]], dtype=numpy.float32)
        XYZ1=XYZ1.T #Transpose
        suv1=P_mtx.dot(XYZ1) #Matrix multiplication
        s=suv1[2,0]
        s_sum += s
s_mean = s_sum / len(corners_u) #Mean scale factor
```

Quindi si è proseguito al calcolo delle posizioni di ogni punto di interesse:

```
#Inverse intrinsic parameters matrix
inv_mtx = numpy.linalg.inv(mtx)
#Inverse rotation matrix
inv_r_matrix = numpy.linalg.inv(rmatrix)
```

```
p_idxs = [0,5,12,31,8] #Image point indexes
calc_points = []
for p in p_idxs:
    #Image point(pixel)
        uv_1 = numpy.array([[
            corners_u[p,0,0],
            corners_u[p,0,1],
            1]], dtype=numpy.float32)
        uv_1 = uv_1.T
        suv_1 = s_mean * uv_1
        xyz_c = inv_mtx.dot(suv_1)
        xyz_c=xyz_c-tvecs[3]
        XYZ=inv_r_matrix.dot(xyz_c)#Real world point
        calc_points.append(XYZ)
```

4 Conclusioni

4.1 Calibrazione singola camera

Considerati i 60 diversi set di immagini analizzati, generati dall'originale tramite gli script in pre-processing, è possibile affermare che il peso dominante rispetto alla calibrazione della singola camera con immagini coerenti allo scopo, è sicuramente riguardante le dimensioni delle immagini.

Infatti considerando una dimensione originale dell'immagine di circa 4000 x 3000 pixel questi in Fig. 1 sono i risultati ottenuti.

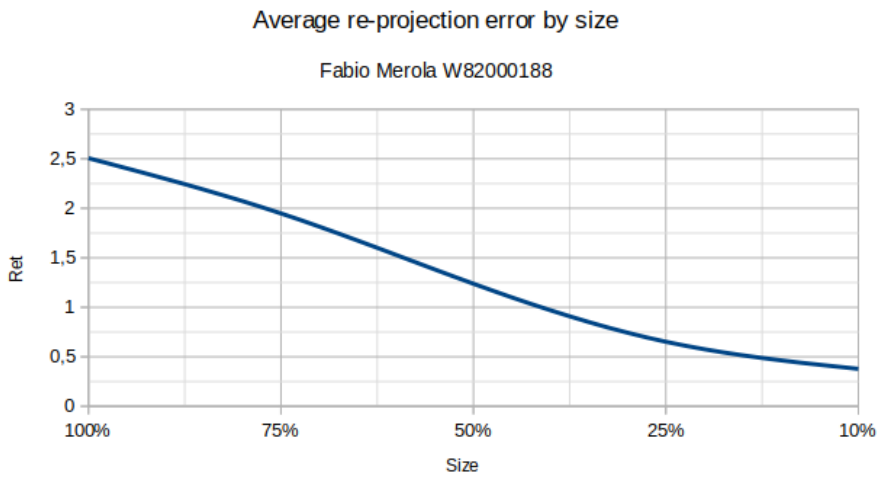

Fig. 1: Errore medio di riproiezione per dimensioni variabili delle immagini.

L'errore di riproiezione nel caso ottimo(10%) è migliore rispetto a quello nel caso peggiore(100%) di circa l'85%.

Un aspetto curioso, riguarda invece le analisi svolte sull'aumento del contrasto che hanno fornito i risultati in Fig. 2.

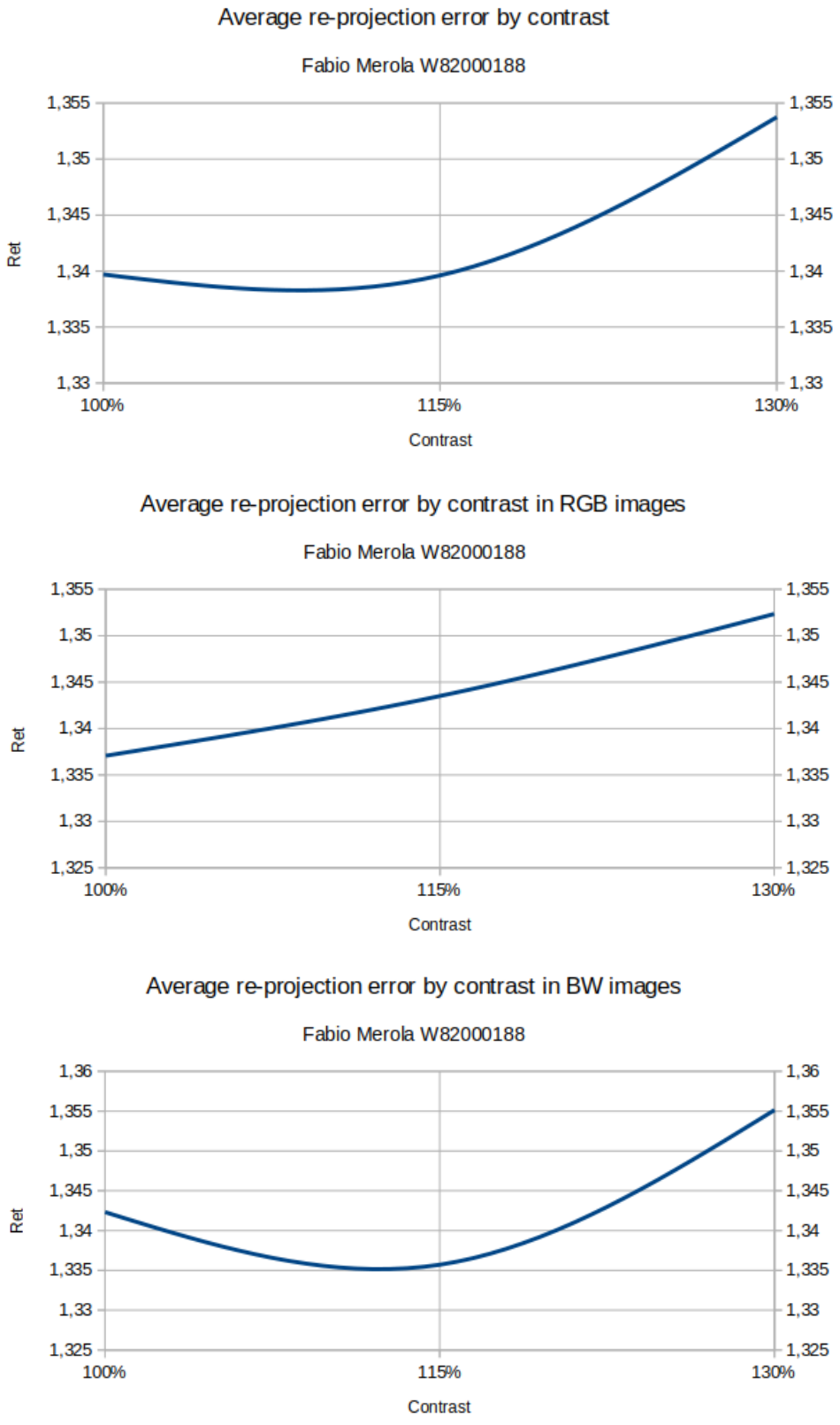

Fig. 2: Errore medio di riproiezione per valore di contrasto variabile.

Gli aspetti interessanti da prendere in considerazione sono due:

- Un aumento di circa il 15% del contrasto ha mediamente effetti positivi solo in immagini in Grayscale, quantificati in un miglioramento dell'errore di riproiezione di circa l'1%. In genere, esclusa questa eccezione, un aumento di contrasto non porta a miglioramenti;

- Contrariamente al trend che appare nei grafici di media, il set che ha reso maggiormente in termini di errore di riproiezione è in RGB, con contrasto aumentato del 15%. Mentre quello che ha reso minormente è in RGB, con aumento nullo del contrasto.

Differenza in media, tra valori d'errore ottenuti con sole immagini RGB rispetto a quelle in Grayscale risulta assolutamente trascurabile.

La comparativa tra i due dispositivi non ha portato a nessuna conclusione degna di nota, per completezza si espongono le medie per dispositivo:

Dispositivo	**Media errore di riproiezione**
SM-G950F	1,34999060532063
LYA-L29	1,33870965388403

A titolo esemplificativo, in Fig. 3, la rettifica di un'immagine presa dal set migliore con stima dei punti della scacchiera sovrapposta.

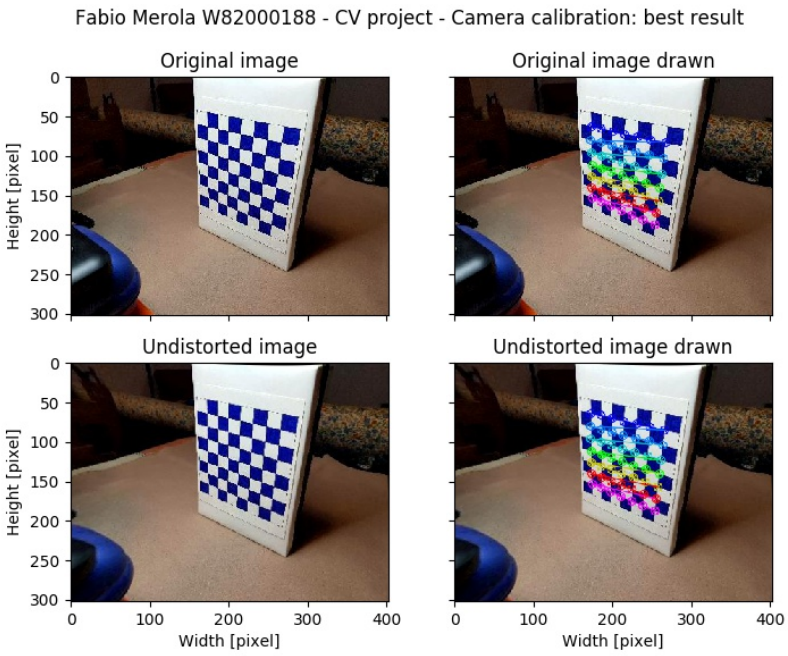

Fig. 3: Calibrazione della camera: set migliore.

4.2 Calibrazione sistema stereoscopico

Dopo le varie analisi fatte sui risultati della calibrazione stereo si evince subito come il problema principale sia dato dalla posizione delle camere. Queste infatti, sono state posizionate ad una distanza di circa 70 centimetri l'un dall'altra, il che ha compromesso la possibilità di ottenere risultati soddisfacenti.
Il test iniziale ha infatti portato a risultati inutilizzabili, riportati in Fig. 4.

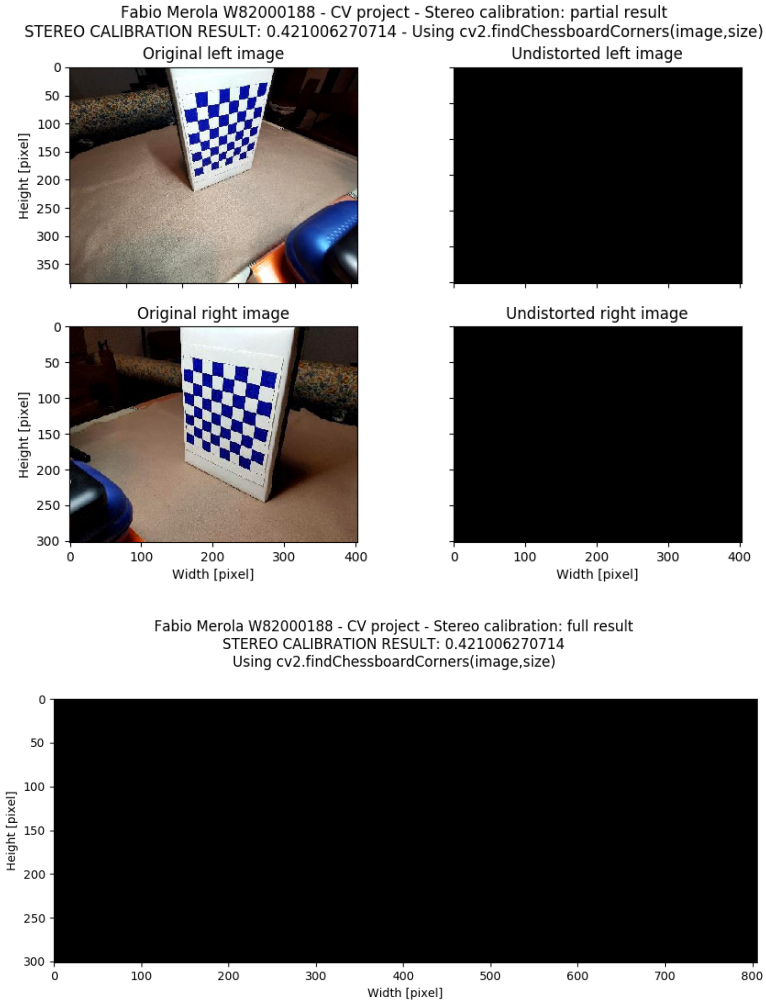

Fig. 4: Calibrazione sistema stereoscopico: risultato peggiore.

Come si evince dalle immagini, nonostante il valore dell'errore di riproiezione sia relativamente basso(circa 0.421), il risultato si presenta come un'immagine totalmente nera.

Si è quindi optato per l'utilizzo di una funzione di rilevazione dei punti della scacchiera più precisa di quella discussa a lezione, nel dettaglio:

```
cv2.findChessboardCornersSB(...)
```

Questa utilizza la trasformata localizzata di Radon, approssimata da filtri box che, non solo permette un rilevamento dei punti più robusto e più rapido, ma riesce anche a ritornare la posizione dei sub-pixel che identificano i punti in maniera più precisa, rispetto alla funzione:

```
cv2.cornerSubPix(...)
```

I risultati palesemente migliori, anche se discutibili, sono riportati in Fig. 5.

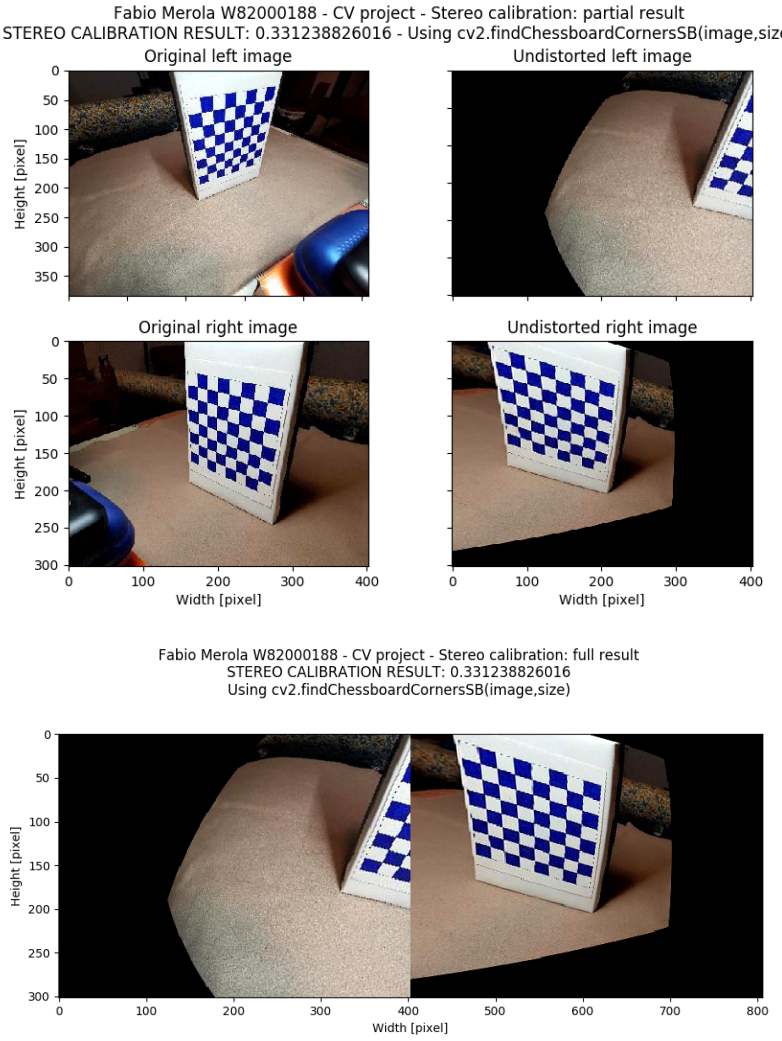

Fig. 5: Calibrazione sistema stereoscopico: risultato migliore.

Il valore dell'errore di riproiezione stavolta ammonta a circa 0.33124, ben migliore del precedente.

4.3 Misurazione oggetto reale

Nonostante il set di immagini di riferimento sia stato scelto randomicamente non ha portato a pessimi risultati. Infatti, come visionabile in Fig. 6, le misurazioni hanno un errore medio del 6.06% rispetto a quelle reali.

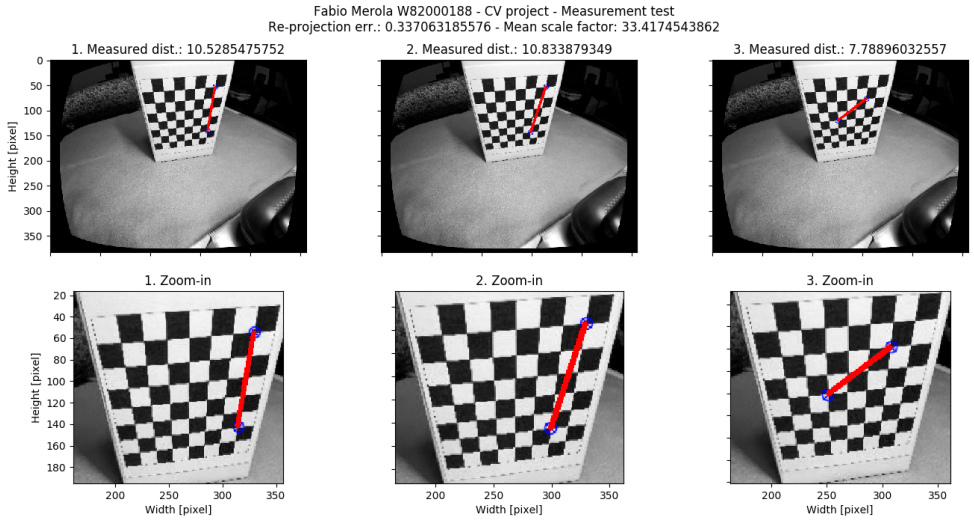

Fig. 6: Misurazione oggetto reale, da sinistra: Mis. 1, Mis. 2, Mis. 3.

Queste, per facilitarne la lettura, le misurazioni:

Mis.	Reale	Calcolata	Errore
1	10.0 cm	10.53 cm	5.3%
2	10.2 cm	10.83 cm	6.18%
3	7.3 cm	7.79 cm	6.71%

4.4 Sviluppi futuri

Sicuramente, per quanto riguarda la calibrazione della singola camera, si può pensare alla sostituzione della funzione di rilevamento punti della scacchiera come fatto per la calibrazione del sistema stereoscopico. Questo consentirà sicuramente di ottenere risultati migliori ma soprattutto in minor tempo.

Per quanto riguarda la calibrazione di un sistema stereoscopico potrebbe invece essere interessante studiare quali posizioni, o quale range di posizioni, siano più adatte per le camere.

Infine, per la misurazione, considerato il notevole peso che il fattore di scala ha nei suoi confronti, si può intraprendere uno studio sulla distribuzione dei valori di questo fattore, al fine di utilizzare l'indice matematico più adatto a riassumere l'insieme in un unico valore.

Riferimenti

[1] Repository GitHub del progetto : Camera-and-stereo-calibration.
[https://github.com/Dophy6/Camera-and-stereo-calibration].
Ultimo accesso: 11 Aprile 2020.

[2] Documentazione OpenCV : `cv2.findChessboardCornersSB(...)`.
[https://docs.opencv.org/4.2.0/d9/d0c/group__calib3d.html].
Ultimo accesso: 11 Aprile 2020.