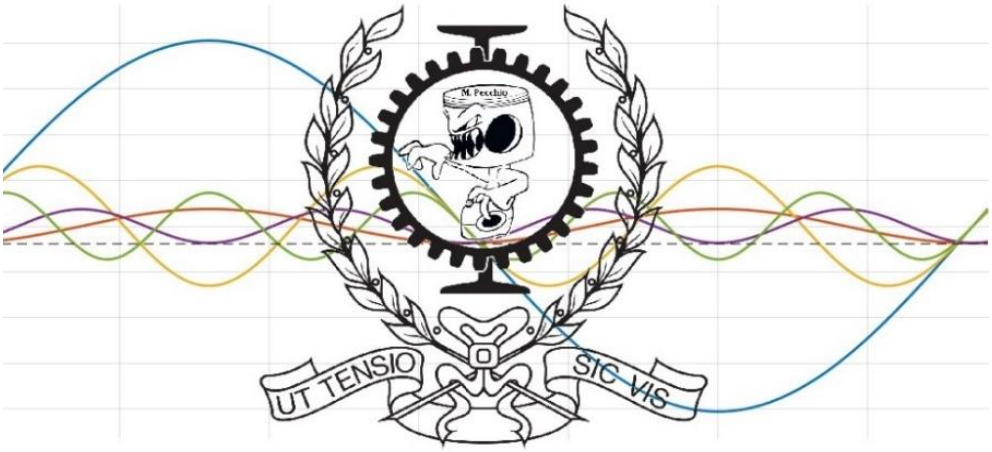
Corso di laurea Magistrale in Ingegneria Meccanica

SISTEMA DI IDENTIFICAZIONE SCENA STRADALE

a.a. 2018-2019

Relatori ---------- : Prof. Ing. *Massimo Sorli*
Prof. Ing. *Stefano Mauro*

Tesista ---------- : *Michele Pecchio*
Ghiringhelli Rota (S 238306)



ABSTRACT

E' esperienza comune ad ogni automobilista che la presenza dei montanti strutturali di tipo "A" possa talvolta compromettere seriamente la visibilità del conducente. La necessità di disporre di una rigidezza strutturale consistente, non permette la semplice rimozione di tali elementi senza sconvolgere il progetto del veicolo. Rassegnatomi di fronte a questo dato di fatto, l'idea che mi è sorta è quella di collocare un display in corrispondenza del montante A per tentare di riprodurre al guidatore l'immagine che egli dovrebbe percepire in assenza del pilastro stesso. La presente trattazione è quindi destinata a valutare gli aspetti tecnici necessari alla realizzazione di questo "*sistema di identificazione scena stradale*".

Uno dei grossi problemi che impone il continuo aggiornamento dell'immagine (oltre che alla velocità relativa tra veicolo e resto del mondo), è la possibilità che il conducente "ruoti" i propri occhi oppure che li sposti a seguito di una variazione di posizione della sua testa.
La seguente tesi tuttavia, si arresta prima di giungere alla discussione degli algoritmi di rilevamento degli occhi (eyes detection), in quanto sono emersi numerosi aspetti che hanno rallentato parecchio lo sviluppo del programma di sintesi-immagine ancora nelle sue fasi intermedie.

Si discuterà dunque soltanto una versione semplificata del problema, in cui si ipotizza che al posto di un'osservatore umano vi sia una fotocamera. Data inoltre l'attuale mancanza di budget, il passaggio dei dati al display non sarà fisicamente realizzato, ci si accontenterà bensì di simularla mediante un grafico animato in Matlab.

Per quanto detto, in questo documento verranno esposti solo i principali step e risultati che hanno portato il progetto allo stato di avanzamento corrente. Si porrà l'accento sui diversi richiami teorici di cui è stato fatto uso, al fine di consentire al lettore di orientarsi e di comprendere sufficientemente a fondo le principali fonti di ritardo nell'ottenimento dell'immagine sintetizzata sul display.

Poiché il progetto del sistema non è ancora stato completamente terminato, buona parte del tempo rimanente è stato investito nel rendere il programma "il più portabile possibile". In altre parole, si sono realizzati diversi script Matlab per consentire agli eventuali interessati di realizzare con il proprio PC il setup sperimentale in un minor tempo possibile.
In questo modo, lo studio del sistema resta potenzialmente ancora aperto. Si rimandano gli eventuali interessati alle conclusioni per capire immediatamente e da un punto di vista più concreto quali sono gli aspetti a cui occorre far fronte per la prosecuzione del progetto.

Michele Pecchio Ghiringhelli Rota

INDICE

INDICE DELLE FIGURE

INDICE DELLE TABELLE

1 – INTRODUZIONE

1.1 – Introduzione al problema

E' esperienza comune ad ogni automobilista, che i montanti strutturali (soprattutto i cosiddetti "A" e "B") spesso ostacolano la visuale stradale, intaccando parzialmente la sicurezza della guida.
La resistenza meccanica del telaio, dev'essere garanita e pertanto non si può pensare di effettuare una semplice eliminazione di tali elementi.
Escludendo come soluzione possibile quella di realizzare un "tetto a sbalzo", l'idea che ci accingiamo a valutare è quella di *ricostruire l'immagine occultata al guidatore per causa dei montanti mediante un* ***sistema di telecamere e display***. L'idea è schematizzata in *Figura 1*:

Figura 1 – Introduzione al problema

Risulta di immediata comprensione il fatto che la zona occultata dipenda in particolar modo dal tipo di vettura, più precisamente dalla posizione relativa tra la testa del pilota (in particolare i suoi occhi) ed i montanti strutturali. Si tenga presente che il concetto di *ricreare un immagine sulla superficie di un corpo che occlude il campo visivo* è evidentemente estensibile anche ad altri contesti dell'automotive, del settore aeronautico, industriale, ecc.

1.2 – Ricerca letteratura esistente

1.2.1 – Metodo di ricerca

Prima di effettuare lo studio del problema ed il successivo progetto da un punto di vista tecnico, si decide di effettuare una ricerca on-line per comprendere se l'idea in questione sia già stata concepita da qualcun altro. In caso positivo, ne verrà valutata ed eventualmente rielaborata la tecnica utilizzata.

Si sceglie di eseguire la ricerca sul web e all'interno di alcune piattaforme private. In quest'ultimo caso, il *Politecnico di Torino* consente a noi studenti l'accesso gratuito a una molteplicità di brevetti, libri e articoli di ogni genere.

Essendo io in questa fase leggermente fuori dal mio specifico campo di studi, da progettista si è reso anzitutto necessario che mi facessi un'idea sulle key-words da utilizzare nella ricerca. Per questa ragione, dopo una preliminare infarinatura sul web, ho deciso di eseguire un'investigazione "più sistematica possibile", generando cioè per ciascuna parola chiave utilizzata nell'indagine, alcuni indici atti a valutarne la potenziale utilità per il successivo sviluppo tecnico. Per meglio comprendere questo concetto, si faccia riferimento alle celle di *Tabella 1*. A seguito del lavoro svolto, ho compreso che le aree ingegneristiche di cui si è fatto maggiormente ricorso vanno sotto il nome di *Digital Image Processing*, *Computer Science* o *Computer Vision*. A tal proposito, si tenga presente che l'ambito della visione è attualmente ancora in forte sviluppo nonchè in continuo aggiornamento.

1.2.2 – Corrispondenze trovate

In *Tabella 1,* il riassunto dell'analisi effettuata. Come chiave di lettura per la tabella di ricerca, si faccia riferimento al seguente specchietto:

LEGENDA	
*BDA	= Banche Dati in Abbonamento
*PEA	= Periodici elettronici in Abbonamento
*EBA	= Electronic book in Abbonamento
/	= La ricerca non ha prodotto risultati/risultati pertinenti'
In colore grigio	= Colonna che sta variando

RICERCHE INTERNET - MACROARGOMENTO: "Display on pillar"								
Area virtuale ricerca	SottoArea virtuale ricerca	Parole chiave inserite	Elementi d'interesse trovati	Inventore	Assegnatario	Attinenza (../10)	Utilità per tesi (../10)	Note
WEB	Corriere.it	Transparent car pillar	Arriva il «montante trasparente»	Jaguar & Land Rover	/	10	4	Dette case correlano il sistema con la cosiddetta ghost navigation
WEB	/	Screen pillar	/	/	/	/	/	/
WEB	/	Display car pillar	Toyota patents an A-pillar you can see through	Toyota	/	9	3	Sistema di tipo differente: ordine 0 grazie agli specchi
WEB	Indiegogo.com	Ninja pillar	Ninja pillar: To make A-pillars transparent	Zen Gasai	/	10	8	/
WEB	/	Invisible car pillar	/	/	/	/	/	/
WEB	/	Camera display car pillar	/	/	/	/	/	/
WEB	/	Camera display vehicle	/	/	/	/	/	/
WEB	/	Non blind car pillar	/	/	/	/	/	/
WEB	/	Diminished reality car	/	/	/	/	/	/
WEB	/	Augmented reality car pillar	/	/	/	/	/	/
WEB	/	Augmented reality vehicle	/	/	/	/	/	/
WEB	/	See through vision	/	/	/	/	/	Nulla correlato al montante auto
BDA*	Orbit intelligence	Transparent pillar	Trasparent A pillar structure	Abe Takeshi	Abe Takeshi	8	2	Idea banale, auto non robusta
BDA	Orbit intelligence	Screen pillar	/	/	/	/	/	/
BDA	Orbit intelligence	Display pillar	/	/	/	/	/	/
BDA	Orbit intelligence	Ninja pillar	/	/	/	/	/	/
BDA	Orbit intelligence	Invisible car pillar	Automobile A-pillar optic invisible device	Rao Zili	Rao Zili	9	5	Idea semplice ed efficace, uso di specchi
BDA	Orbit intelligence	Camera less pillar	Driving assist device	Yshiguro Yoko	Denso, Toyota Motor	10	8	/
BDA	Orbit intelligence	Camera display car pillar	Non-blind area display device for car A pillar	Lian Desheng	Lian Desheng	9	7	/
BDA	Orbit intelligence	Camera display car pillar	Automobile A-pillar perspective vehicle-mounted display device	Gha Zehnhai	Jilin university	10	9	Identico a quanto da me immaginato
BDA	Astm Compass	Transparent pillar	/	/	/	/	/	/
BDA	Astm Compass	Trasparent A pillar structure	/	/	/	/	/	/
BDA	Astm Compass	Screen car pillar	/	/	/	/	/	/
BDA	Astm Compass	Invisible car pillar	/	/	/	/	/	/
BDA	Astm Compass	Camera display car pillar	/	/	/	/	/	/
BDA	DAAI	Transparent pillar	/	/	/	/	/	/
BDA	DAAI	Ninja pillar	/	/	/	/	/	/
BDA	DAAI	Invisible car pillar	/	/	/	/	/	/
BDA	Scopus	Car pillar	Development of a detection road users system in vehicle A-pillar blind spots	Beresnev P.O.	/	9	5	Questa piattaforma non da dati tecnici. Keywords apprese: control system of blind spots
BDA	Web of science	Transparent car pillar	A real-time augmented view synthesis system for transparent car pillars	Chang	/	7	1	/
BDA	Web of science	Display car pillar	/	/	/	/	/	/
PEA*	ACM digital library	Transparent car pillar	/	/	/	/	/	Piattaforma con materiale
PEA	ACM digital library	Camera display car pillar	/	/	/	/	/	/
PEA	ACM digital library	Car pillar blind	/	/	/	/	/	/
PEA	ACM digital library	Screen car pillar	/	/	/	/	/	/
PEA	EmeraldInsight	Non blind area car pillar	/	/	/	/	/	/
PEA	EmeraldInsight	Camera display car pillar	/	/	/	/	/	/
PEA	EmeraldInsight	Transparent car pillar	/	/	/	/	/	/
PEA	IEEE Xplore	Camera display car	/	/	/	/	/	/
PEA	IEEE Xplore	Transparent car pillar	A real-time augmented view synthesis system for transparent car pillars	Yu-Lin Chang	Graduate Institute of Electronics Engineering	10	9,5	Keywords imparate: View synthesis, Augmented panorama, See through pillar
PEA	IEEE Xplore	See through pillar	Intelligent Surveillance System with See-Through Technology	Yu-Chen Lin*	/	9	9,5	/
PEA	IEEE Xplore	Augmented view camera	An Instant See-Through Vision System Using a Wide Field-of-View Camera and a 3D-Lidar	Kei Oishi	/	9	9,5	/
PEA	IEEE Xplore	Diminished reality	First Deployment of Diminished Reality for Anatomy Education	Naoto Lenaga	/	8	9	Keyword : Camera link model, legare 2 immagini da 2 diverse Videocamere
PEA	INFORMS PubsOnLine Suite	Transparent car pillar	/	/	/	/	/	/
PEA	INFORMS PubsOnLine Suite	Diminished reality	/	/	/	/	/	/
PEA	INFORMS PubsOnLine Suite	Display pillar	/	/	/	/	/	/
PEA	INFORMS PubsOnLine Suite	See through pillar	/	/	/	/	/	/
PEA	INFORMS PubsOnLine Suite	See through	/	/	/	/	/	/
PEA	IOP Journals	Transparent car pillar	/	/	/	/	/	/
PEA	IOP Journals	Diminished reality display camera	/	/	/	/	/	/
PEA	IOP Journals	Camera display car pillar	Development of a detection road users system in vehicle A-pillar blind spots	P O Beresnev	/	9	9	Display integrato nel montante A
PEA	IOP Journals	Display blind spot car	/	/	/	/	/	/
PEA	OSA Optics infobase	Camera display car pillar	/	/	/	/	/	Periodici di ottica. Keyword: head worn display
PEA	OSA Optics infobase	See through vision	/	/	/	/	/	/
PEA	OSA Optics infobase	Diminished reality	/	/	/	/	/	/
PEA	OSA Optics infobase	Non blind area	/	/	/	/	/	/

PEA	PNAS	Invisible car pillar	/	/	/	/	/	/
PEA	PNAS	Display pillar	/	/	/	/	/	/
PEA	PNAS	Transparent car pillar	/	/	/	/	/	/
PEA	PNAS	Non blind car area	/	/	/	/	/	/
PEA	SAE MOBILUS	See through vision	Automotive Vision & Obstruction Assessment For Driver	S Lakshimi	/	8	9	Interessante, piattaforma dell'automotive
PEA	SAE MOBILUS	Transparent pillar	AION A-Pillars: Enhancing Passenger Safety and Driver Visibility	Shweta Rawat	/	8	9	Keyword appresa: Head up syntetic vision display
PEA	SAE MOBILUS	Screen pillar	/	/	/	/	/	/
PEA	SAE MOBILUS	Display car pillar	/	/	/	/	/	/
PEA	SAE MOBILUS	Ninja pillar	/	/	/	/	/	/
PEA	SAE MOBILUS	Invisible car pillar	Design and Implementation of an Image-Based Driver Attention Warning System on Low-Cost DSP Platform	Yu-Sheng Liao	ARTC	7	9	Idea: face detection con più camere
PEA	SAE MOBILUS	Invisible car pillar	Development of All-Around View Image Processing Method for Approaching Vehicle Using Dual Adaptive Camera System	Nonnwats Anantpal	/	7	8,5	Info su fusione di immagini
PEA	SAE MOBILUS	Camera display car pillar	NEW VISION @ GENTEX	/	/	/	/	Sembra interessante. Ma, come tanti altri, è privato
PEA	SAE MOBILUS	Camera display car pillar	Panoramic Electronic Rear Vision for Automotive Applications	Rich Hicks	/	7	9	Utile per schemi funzionali
PEA	Science	Transparent pillar	/	/	/	/	/	/
PEA	Science	Screen pillar	/	/	/	/	/	/
PEA	Science	Display car pillar	/	/	/	/	/	/
PEA	Science	Ninja pillar	/	/	/	/	/	/
PEA	Science	Invisible car pillar	/	/	/	/	/	/
PEA	Science	Camera display car pillar	/	/	/	/	/	/
PEA	Science	No blind car pillar	/	/	/	/	/	/
PEA	Science	Diminished reality car	/	/	/	/	/	/
PEA	ScienceDirect	Transparent pillar	/	/	/	/	/	/
PEA	ScienceDirect	Screen pillar	/	/	/	/	/	/
PEA	ScienceDirect	Display car pillar	/	/	/	/	/	/
PEA	ScienceDirect	Ninja car pillar	/	/	/	/	/	/
PEA	ScienceDirect	Invisible car pillar	/	/	/	/	/	/
PEA	Springer	Transparent car pillar	/	/	/	/	/	Piattaforma molto ricca di libri scientifici (a pagamento) ma con ricerca poco affinabile
PEA	Springer	Display car pillar	/	/	/	/	/	/
PEA	Springer	Ninja car pillar	/	/	/	/	/	/
PEA	Springer	Diminished reality car	/	/	/	/	/	/
EBA*	EBook Central	Transparent car pillar	/	/	/	/	/	Bel servizio, ma utile più a livello teorico che per idee specifiche
EBA	EBook Central	Screen pillar	/	/	/	/	/	Probabilmente si troverebbe qualcosa, ma non specifico per le macchine
EBA	EBook Central	Display car pillar	/	/	/	/	/	/
EBA	EBook Central	Ninja pillar	/	/	/	/	/	/
EBA	EBook Central	Invisible car pillar	/	/	/	/	/	/
EBA	EBook Central	Camera display car pillar	/	/	/	/	/	/
EBA	EBook Central	No blind car pillar	/	/	/	/	/	/
EBA	EBook Central	Diminished reality car	/	/	/	/	/	/
EBA	ELSEVIER e-books	Transparent car pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	Screen pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	Display car pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	Ninja pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	Invisible car pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	Camera display car pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	No blind car pillar	/	/	/	/	/	/
EBA	ELSEVIER e-books	Diminished reality	/	/	/	/	/	/
EBA	ELSEVIER e-books	Augmented reality	/	/	/	/	/	Libri con detto titolo, ma a pagamento
EBA	WILEY	Transparent car pillar	/	/	/	/	/	/
EBA	WILEY	Screen pillar	/	/	/	/	/	/
EBA	WILEY	Display car pillar	/	/	/	/	/	/
EBA	WILEY	Ninja pillar	/	/	/	/	/	/
EBA	WILEY	Invisible car pillar	/	/	/	/	/	/
EBA	WILEY	Camera display car	/	/	/	/	/	/
EBA	WILEY	Camera display vehicle	/	/	/	/	/	/
EBA	WILEY	No blind car	/	/	/	/	/	/
EBA	WILEY	Diminished reality car	/	/	/	/	/	/
EBA	WILEY	Augmented reality	Classification of illumination methods for mixed reality	Katrien Jacobs	/	7	8	/
EBA	WILEY	Augmented reality vehicle	A Mobile Augmented Reality System to Support Machinery Operations in Scholar Environments	ALEJANDRO MONROY REYES	/	7	8,5	Keyword : image fusion
EBA	WILEY	See through vision	Recent Advances in Head-Mounted Light Field Displays	Hong Hua	/	7,5	8,5	

EBA	KNOVEL e-books	Transparent car pillar	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Screen pillar	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Display car pillar	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Ninja pillar	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Invisible car pillar	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Camera display car	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Camera display vehicle	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	No blind spots	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Diminished reality car	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Augmented reality	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	Augmented reality vehicle	/	/	/	/	/	/	/	/	/	/
EBA	KNOVEL e-books	See through vision	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Transparent car pillar	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Screen pillar	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Display car pillar	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Ninja pillar	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Invisible car pillar	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Camera display car	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Camera display vehicle	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	No blind spots	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	Diminished reality car	A survey of diminished reality: Techniques for visually concealing, eliminating, and seeing through real objects	/	Shohei Mori	/	8	9	Keyword : Seeing through real object, Camera pose estimation, Users point of view tracking	/	/	/
EBA	LECTURE NOTES	Epipolar geometry	Discrete epipolar geometry	/	Masatoshi Hamanaka	/	6	9	Epipolar geometry, Image matching in stereo, Calibration of stereo-digital image correlation	/	/	/
EBA	LECTURE NOTES	Augmented reality vehicle	/	/	/	/	/	/	/	/	/	/
EBA	LECTURE NOTES	See through vision	A method of creating a transparent space based on mixed reality	/	Sunao Hashimoto	/	6	9	Keyword : Mixed reality	/	/	/
EBA	REFEREX Engineering	Transparent car pillar	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Screen pillar	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Display car pillar	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Ninja pillar	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Invisible car pillar	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Camera display car	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Camera display vehicle	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	No blind spots	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Diminished reality car	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Epipolar geometry	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	Augmented reality vehicle	/	/	/	/	/	/	/	/	/	/
EBA	REFEREX Engineering	See through vision	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Transparent car pillar	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Screen pillar	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Display car pillar	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Ninja pillar	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Invisible car pillar	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Camera display car	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Camera display vehicle	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Non blind car	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Diminished reality car	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Epipolar geometry	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	Augmented reality vehicle	/	/	/	/	/	/	/	/	/	/
EBA	SPRINGER e-books	See through vision	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Transparent car pillar	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Screen pillar	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Display car pillar	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Ninja pillar	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Invisible car pillar	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Camera display car	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Camera display vehicle	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Non blind car	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Diminished reality car	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Epipolar geometry	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	Augmented reality vehicle	/	/	/	/	/	/	/	/	/	/
EBA	Wiley Online Library	See through vision	/	/	/	/	/	/	/	/	/	/

Tabella 1 – Indagine letteratura esistente

2 – LETTERATURA - ARTICOLI PIU' RILEVANTI

Si tenga presente che la traduzione dei successivi articoli è stata aricchita con l'aggiunta di ulteriori commenti atti ad agevolarne la comprensione. Trattandosi infatti di brevi articoli, spesso gli autori hanno dato per scontato che il lettore abbia già una conoscienza regressa consistente e sufficiente a comprendere quanto esposto. Laddove il testo originale è stato integrato con informazioni extra, sono state poste delle parentesi quadre. In altre parole, la logica utilizzata è del tipo:

Spezzone della traduzione del testo originale dell'articolo, [CommentiAggiuntiDaMe], prosecuzione della traduzione dell'articolo.

2.1 – La bozza di Jaguar & Land Rover

L'esistenza del problema è stata riscontrata già da diversi anni, e la bozza di soluzione del tipo "con display" più recente reperibile sui classici siti web risale al 2014. In quell'anno, le marche automobilistiche citate hanno annunciato l'idea, ma attualmente in mercato non si trovano ancora vetture con detta soluzione integrata. L'articolo web propone l'immagine di *Figura 2*:

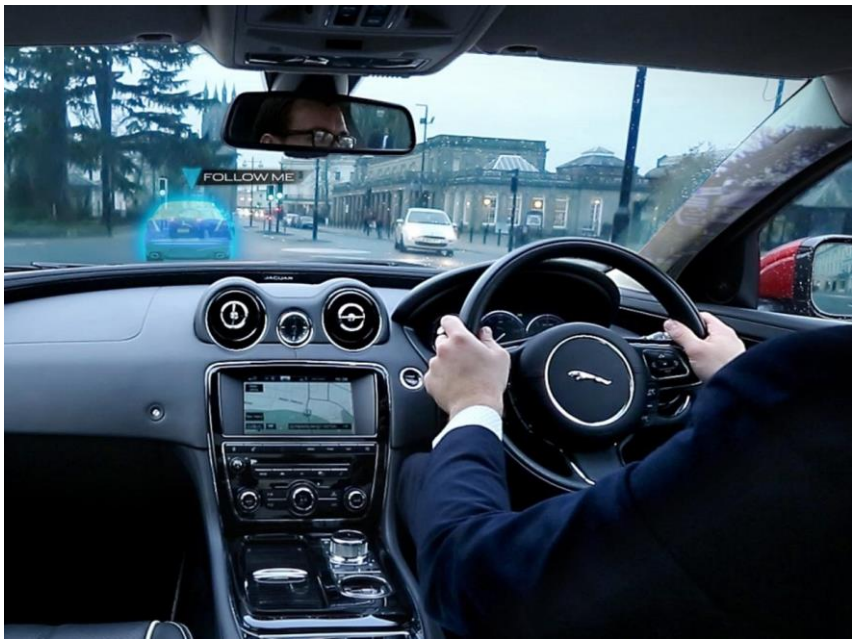


Figura 2 – Jaguar con ricostruzione virtuale della scena occultata dal montante

Si noti che con molta probabilità si tratta di una semplice post-elaborazione della fotografia tramite computer. E' comunque chiaramente comprensibile l'intento di eliminare (o comunque minimizzare) la visuale cieca mediante l'uso di proiettore/display correlato ai montanti A. Jaguar & Land Rover, corredano al sistema anche la possibilità di proiettare sul parabrezza informazioni ritenute di ausilio alla guida. Il sistema venne battezzato "***360 Virtual Urban Windscreen***".

2.2 – Soluzione proposta da Toyota

Più suggestiva è l'idea proposta da Toyota. Digitando sui motori di ricerca le parole chiave "**Trasparent A Pillar**", le pagine web trovate non sono ("fortunatamente") molte, ma tra di esse emerge quella contenente un articolo in cui si discute di una soluzione alternativa. Trattasi in questo caso di un sistema di specchi che riesce ad "aggirare" i montanti, e "trasportare" l'immagine all'interno dell'abitacolo, sui montanti stessi.
In ogni caso, l'idea proposta da Toyota è stata brevettata ma il mercato (stando alle rapide ricerche su internet) non ha ancora commercializzato auto corredate del sistema illustrato. Lo schema rappresentativo è illustrato in *Figura 3*:

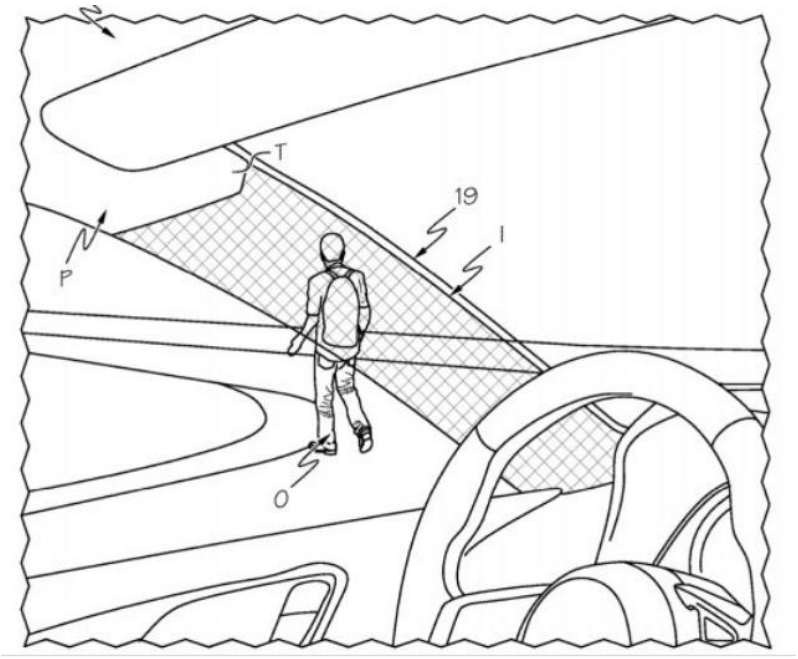


Figura 3 – Camera-less transparent A-pillars

Il sistema in esame venne chiamato "***Camera-less transparent A-pillars"*** . La pagina web risale a poco più di un anno fa: il 22/08/2017.

2.3 – A real-time augmented view synthesis for transparent car pillars

2.3.1 – Abstract

In questo documento, verrà discussa la realizzazione di un sistema di "Visione sintetizzata virtualmente aumentata in tempo reale". Grazie alla particolare stima della profondità mediante sistema di telecamere trinoculare, la ricostruzione delle immagini diviene abbastanza rapida da consentire la riproduzione dell'immagine in Real-time. La potenza di calcolo richiesta non è eccessiva, pertanto la realizzazione di tale sistema è possibile mediante computer di qualsiasi fascia. In questo modo, il pilota è in grado di percepire all'interno della sua vettura, la (quasi) totale scena esterna

2.3.2 – Introduction

Al fine di diminuire il tempo di calcolo [per la stima dell'ambiente tridimensionale all'esterno dell'auto (operazione necessaria per la successiva sintesi dell'immagine)], si opta per ridurre il contenuto di informazioni immagazzinate dalle fotocamere. Questo consentirà la successiva riproduzione in Real-Time.
Uno dei problemi facilmente intuibili per questo tipo di sistema, è che l'angolo di veduta di ogni guidatore differisce. Per questa ragione, non è immediato proiettare sul montante A la corretta immagine che dovrebbe percepire il guidatore dal suo punto di vista. Un'idea che è stata pensata, consiste nell'uso della tecnologia a "Proiezione Retro-Riflessiva". Essa consente il cambio di visuale in base al cambio dell'angolo da cui si sta osservando.

2.3.3 – Prior arts

La *vista sintetizzata*, conosciuta anche come *vista interpolata*, consiste di due procedure:

- Trovare la corrispondenza dei Pixel [tra le immagini acquisite da 2 o più dispositivi]
- Interpolare la corrispondenza [riempire cioè le eventuali corrispondenze non trovate]

La maggior parte degli algoritmi di sintesi visiva richiedono la "*Stima della disparità*" [operazione che consiste nel trovare *quanto un determinato punto catturato da una certa fotocamera sia spostato rispetto al corrispettivo punto catturato da un'altra camera*] per la determinazione della corrispondenza dei pixel [tra le immagini catturate da differenti fotocamere]. Tale tema è tipico dell'argomento che va sotto il nome di "*Stereo Matching*" e dell'area della "*Codifica Video*".
La *Stima delle disparità* (*Disparity Estimation*) è divisa in due classi:

- Block-based Disparity estimation
- Dense disparity estimation

La stima della disparità "per blocchi" è tipicamente adottata nei sistemi di codifica a veduta multipla.
La stima della disparità a livello più profondo (*pixel per pixel* [in questo senso prende il nome di "pesante", "Dense")], è maggiormente utile per generare una mappa approfondita di informazioni sui pixel.

Per interpolare la corrispondenza dei pixel, in questo progetto verrà adottato il metodo "*Depth Image Based Rendering*" (*DIBR*). I tradizionali metodi *DIBR* consistono di algoritmi di Image-shifting. Come sempre, l'algoritmo di traslazione di immagini *DIBR* si scontra con il problema dello riempimento dei buchi. [Tali buchi, come vedremo nel seguito, sono figli di una parziale e non totale stima delle disparità.]

Di seguito un diagramma a blocchi per la sintesi dell'immagine sul montante dell'auto:

Fig. 1. The Proposed View Synthesis System block diagram for transparent car pillar

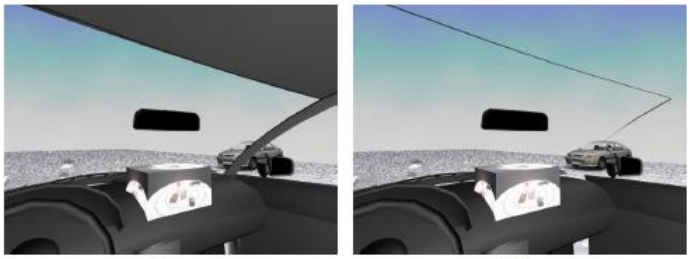


Fig. 2. Simulated Transparent Car Pillar Scenario (a) Original look out scene with A pillar (b) Look out scene with transparent A pillar

Figura 4 – Diagramma a blocchi, articolo R-TAVSSFTCP

In questo articolo, **verrà proposta la sintesi di un sistema di visione attraverso oggetti, Real-Time, ma mantenendo l'angolo d'osservazione del pilota costante**.

2.3.4 – Proposed augmented view synthesis system

Come mostrato in *Figura 4*, è stato utilizzato come input una configurazione trinoculare di videocamere, una stima di profondità trinoculare per la generazione della *depth map* [mappa delle profondità], una trasformazione di veduta [semplice operazione matematico-geometrica] per trasformare la veduta video originale in una vista arbitraria, infine un Rendering per realizzare la tessitura [in senso di riempimento] della nuova immagine sintetizzata secondo il nuovo punto di vista.

Sintetizzare una nuova veduta richiede l'acquisizione della vista originale per concepire quella che poi sarà la vista secondo il nuovo punto d'osservazione. Migliore è la veduta originaria, migliore sarà la predizione [cioè l'immagine ricostruita sulla base della vista di partenza].
Il sistema proposto per il montante delle auto, lavora come esposto in *Figura 5*. Si utilizza un sistrema di telecamere trinoculare (Camera 1, 2 e 3) per la stima della profondità. Il sintetizzatore trasforma la vista ed effettua un Rendering per concepire la generazione del video dal punto di vista del guidatore. Terminata questa fase, il proiettore proietta il video sul montante. In questo

modo, il guidatore avrà la sensazione di vedere attraverso il montante dell'auto. Se il video generato non coincide con il punto d'osservazione del guidatore, possono essere effettuati aggiustamenti in merito al monitoraggio dello sguardo o aggiustamento sulla riproiezione, proprio come ad espempio la semplice regolazione dello specchietto retrovisore.

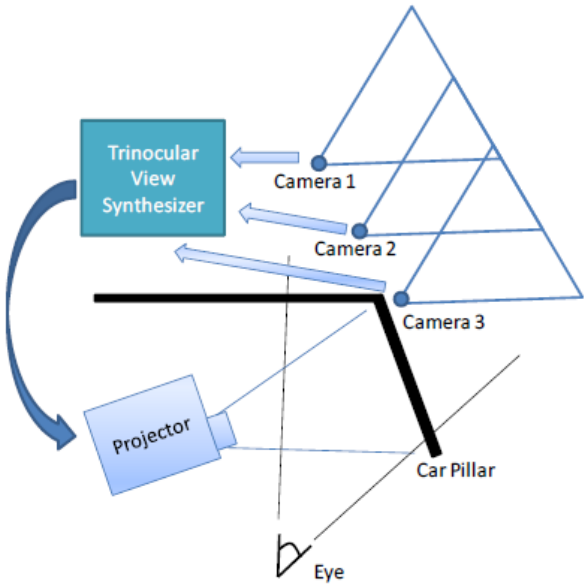


Figura 5 – Diagramma funzionale del montante trasparente, articolo R-TAVSSFTCP

2.3.5 – Trinocular depth estimation

Il diagramma di flusso di *Figura 6* descrive che la stima della profondità passa attraverso la preliminare stima dei: parametri di calibrazione intrinseci alla camera, stima della disparità trinoculare punto per punto, parametri di calibrazione della camera esterni, stima della profondità, profondità di fusione. La stima della disparità trinoculare richiede molta potenza di calcolo. É stato perciò adottato un algoritmo di stima punto per punto abbastanza rapido, frutto del lavoro di *Tsai*. É stata inoltre utilizzata la proprietà di simmetria trinoculare per effettuare delle correzioni alla mappa delle disparità così ottenuta. Come criterio di confronto [tra le immagini acquisite dalle differenti camere] è stata utilizzata la Somma delle assolute differenze, *SAD*. Esso è espresso dalla seguente equazione:

$$SAD_m = \sum_{i,j \in MB} \left(\left| f\left(i - M\,V_{kx}, j - M\,V_{ky}, n-1\right) - f(i,j,n) \right| + \left| f\left(i + M\,V_{kx}, j + M\,V_{ky}, n+1\right) - f(i,j,n) \right| \right)$$

Dove la funzione $f(i,j,n)$ denota l'intensità dell'immagine di input.
i e *j* sono rispettivamente l'asse orizzontale e verticale, mentre il pedice *n* descrive il numero della camera. Il calcolo del primo *SAD* descrive la relazione tra la camera di sinistra e quella centrale, mentre il secondo *SAD* descrive quella tra la centrale e quella di destra. Questa proprietà di simmetria trinoculare aiuta a trovare i vettori di disparità in modo maggiormente accurato.

Combinando tale proprietà con il meccanismo di ricerca veloce, è possibile raggiungere il processamento in Real-Time. Per accelerare il processo di confronto, è stato utilizzato l'algoritmo di ricerca rapida 1D.
La mappa delle disparità è convertita in mappa di profondità attraverso la triangolazione a partire dai vettori di disparità [ottenuti a seguito dell'applicazione del criterio SAD alle immagini] e parametri esterni alla camera. La stima delle disparità pixel per pixel e la conversione della mappa di profondità è effettuata come da *Figura 6*. Questo step fornisce dunque il sistema di "sintesi visiva" con la mappa di profondità che occorre ai successivi step.

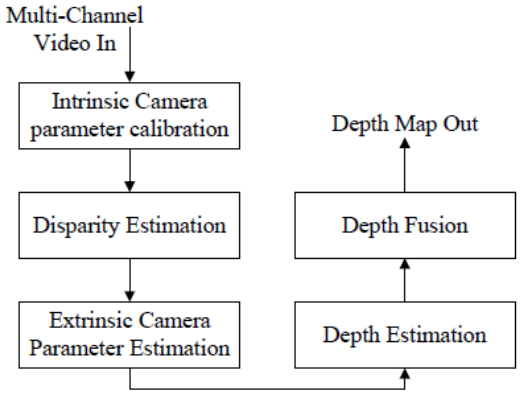


Figura 6 – Flow chart della Stima trinoculare della profondità, articolo R-TAVSSFTCP

2.3.6 – View transform

Dopo aver ottenuto ad output la mappa di profondità a partire dalla stima trinoculare della profondità, **è stata adottata la trasformazione geometrica del cambio di vista combinata con informazioni sulla profondità per costruire la relazione tra la posizione della camera e l'occhio dell'osservatore**. In *Figura 7*, le veci del guidatore sono fatte da una seconda camera (E) dotata di nuova posizione e differenti parametri (lunghezza focale, taglia del piano immagine, ecc). Per trasformare ulteriormente l'oggetto reale [proiettandolo] sul piano immagine della seconda camera, il vettore traslazione dalla camera all'occhio concorre al completamento della matrice di traslazione $\vec{T} = \overrightarrow{C_1} - \vec{E}$.
Il "mappaggio" dei punti degli oggetti tra i diversi piani immagini [associati ciascuno ad una diversa camera] è descritto dalle equazioni sottostanti. Detti:

- P_1, Proiezione della posizione dell'oggetto nella camera C_1
- M_1, Matrice di proiezione della camera C_1
- P_E, Posizione dell'oggetto proiettato nella camera E [che fa le veci del guidatore]
- M_E, Matrice di proiezione della fotocamera-guidatore

Il mapping da P_1 a P_E può essere ottenuto sostituendo l'ultima espressione nella seguente:

$$P_1 = M_1 \left(\vec{P} - \overrightarrow{C_1}\right)$$

$$P_E = M_E \left(\vec{P} - \vec{E}\right)$$

$$M_1^{-1} P_1 + \overrightarrow{C_1} = \vec{P}$$

$$P_E = M_E \left(M_1^{-1} P_1 + \overrightarrow{C_1} - \vec{E}\right) = M_E \left(M_1^{-1} P_1 + \vec{T}\right)$$

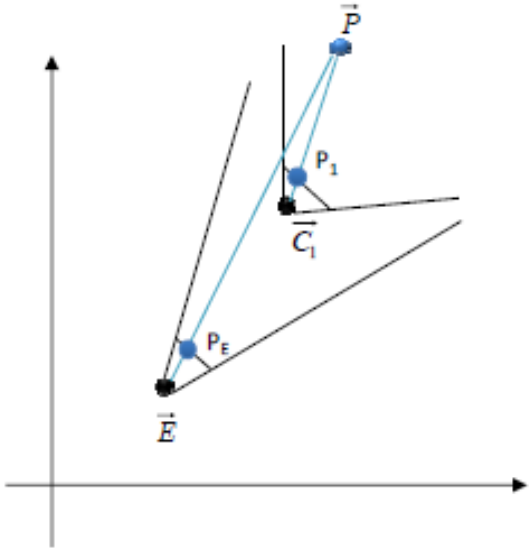


Figura 7 – Relazioni tra posizione camera ed occhio E del guidatore, articolo R-TAVSSFTCP

2.3.7 – Texture-mapping Free Viewpoint Depth Image Based Rendering

Il "*Free Viewpoint Depth Image Based Rendering*" (*FVDIBR*) proposto, utilizza il GPU nei computers per la valutazione in Real-Time. Il tessuto della mappa *FVDIBR* proposto, è descritto in *Figura 8*. I dati di input sono l'immagine vista dal centro e la mappa di profondità. Anzitutto, occorre convertire la mappa di profondità nei vertici con valori in Z (immagine a griglia d i*Figura 8*). Qualsiasi punto dell'immagine originale sarà assegnatario di una certo valore Z, in aggiunta al valore di colore di quel punto nell'immagine iniziale. Dopo la poligonazione, il processo di rendering è assegnato alla GPU. Quest'ultima guarda l'intera immagine come immagine planare. Così, i "buchi" introdotti dal cambiamento di profondità stanno per essere compensati da un metodo coinvolgente la loro ombreggiatura.

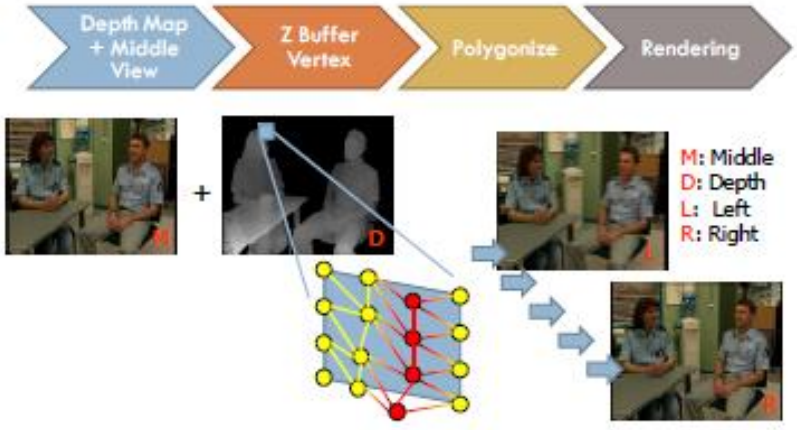


Figura 8 – Rillievo punti basato su algoritmo Deep e Rendering, articolo R-TAVSSFTCP

Dopo la *FVDIBR*, il video che dovrebbe essere visto punto di osservazione del pilota è consegnato al proiettore. Così facendo, la sintesi dell'immagini è effettuata e il guidatore setta semplicemente alcuni parametri di trasformazione [occorrenti all'algoritmo di sintesi dell'immagine] in base alle sue preferenze.

2.3.8 – Augmented Subjective View Results

É stato fissato un pannello bianco rappresentativo del montante all'interno dell'auto. In *Figura 9*, un uomo si sta nascondendo dietro il montante "virtuale". Da un punto d'osservazione qualsiasi, il sistema non può chiaramente riprodurre l'immagine corretta. Le rimanenti 3 immagini mostrano invece la visione dal punto di vista della telecamera, ben ricostruita.

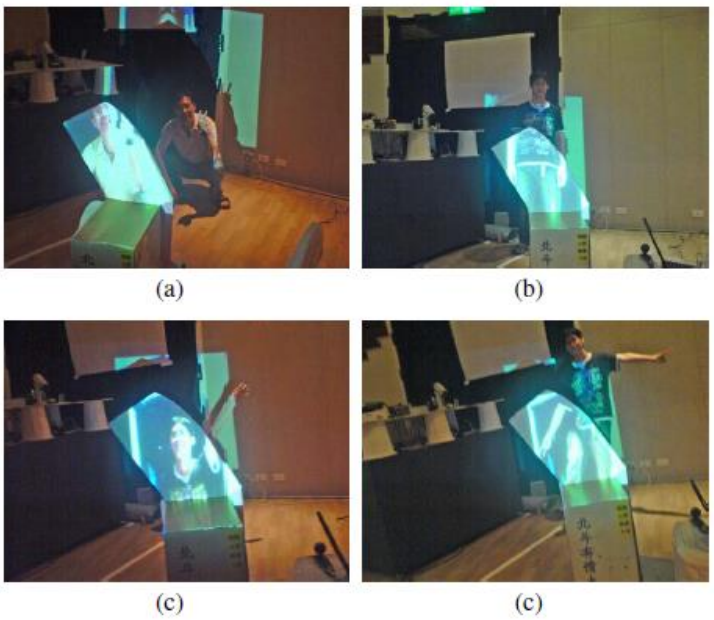


Figura 9 – Real-Time experimental results, articolo R-TAVSSFTCP

2.4 – An instant See-through Vision System Using a Wide Field-of-view Camera and a 3D-Lidar

2.4.1 – Abstract

La "*Realtà Diminuita*" *(Diminished Reality*), ci rende possibile vedere attraverso oggetti reali che occludono aree nel nostro campo visivo. In questo articolo saranno esaminati i due principali problemi sottesi allo sviluppo della "visione attraverso oggetti" in Real-Time. Anzi tutto, ocorre sottolineare che la visione attraverso gli oggetti richiede un'area comune per poter calibrare ogni camera utilizzata nell'ambiente applicativo. Un ulteriore problema da considerare è che il campo visivo è limitato, e molti approcci sono limitati dal tempo di calcolo, sensoristica, utilizzo di Marker fiduciali. Infine, le applicazioni di "visione attraverso" assumono che lo sfondo sia piano per favorire l'allineamento delle immagini. Verrà perciò proposto un "sistema di visione attraverso" [molto più immediato, del tipo] "piazza e utilizza" che sfrutta una singola camera RGB-D ad ampia veduta ed un rilevatore 3D. Questa strumentazione consentirà il calcolo Real-Time e senza l'esecuzione di una pre-calibrazione tra l'ambiente e le camere utilizzate.
In particolare, verranno discussi:

- Sistema istantaneo di vista attraverso oggetti non richiedente la calibrazione tra ambiente e le camere, eccetto per il settaggio della camera RGB-D ad ampio campo visivo.
- Confronto tra le caratteristiche dello sfondo visto da una camera "ad occhio di pesce" (Fisheye) e una camera pinhole [ovvero modellizzabile come se il suo obiettivo fosse puntiforme. Conseguentemente, dotata di immagini non distorte dall'eventuale convessità della lente]
- Dimostrazione di accuratezza, robustezza ed applicabilità utilizzando video in Real-Time catturanti scene indoor ed outdoor

2.4.2 – Related Work

Kameda e la sua équipe proposero l'uso di telecamere di sorveglianza calibrate in anticipo come camere per la visione di zone nascoste. Il metodo consentiva la visualizzazione di uno spazio nascosto dall'edificio per mezzo della conversione delle immagini ottenute dalle camere in immagini percepite dall'utente. Nella conversione delle immagini, si assunse che l'area da visualizzare era piana, o sufficientemente lontana da essere assunta come tale. In questo metodo, **fu utilizzato un modello CAD** per visualizzare le parti nascoste e per stimare la posizione della camera utente dalle camere d'osservazione. Come noto, non è pratico disporre un modello CAD dell'ambiente e la relativa generazione è time-consuming.

Barnum e la sua équipe acquisirono una regione della vista comune tra un utente ed un osservatore dalla vista occultata, mediante una camera aggiuntiva che osservava entrambe dette regioni. Anche questo metodo assunse camere pre-calibrate ed uno sfondo piano durante la trasformazione delle immmagini di sfondo in immagini percepite dall'utente.

Tsuda e la sua équipe, utilizzarono dei marker fiduciali posizionati sul muro per favorire la stima della posizione della camera.

Tutti i metodi esposti pocanzi stimano le posizioni dell'utente online [online in senso di Real-Time, e non preventivamente], ma assumono che tutti gli sfondi delle camere e le camere siano preventivamente calibrate [calibrazione eseguita offline, effettuata cioè una sola volta e a monte della sintesi dell'immagine che invece sarà in Real-Time].

In questo articolo è stata acquisita una regione visiva comune all'osservatore avente visione occultata e la camera ad ampio campo visivo RGB-D. In altre parole, la camera ad occhio di pesce consente una corrispondenza diretta tra le camere, ed un rilevatore laser 3D consente la ricostruzione della struttura in sfondo. In *Figura 10* è schematicamente descritto il numero di trasformazioni di coordinate online e offline dei vari metodi esposti:

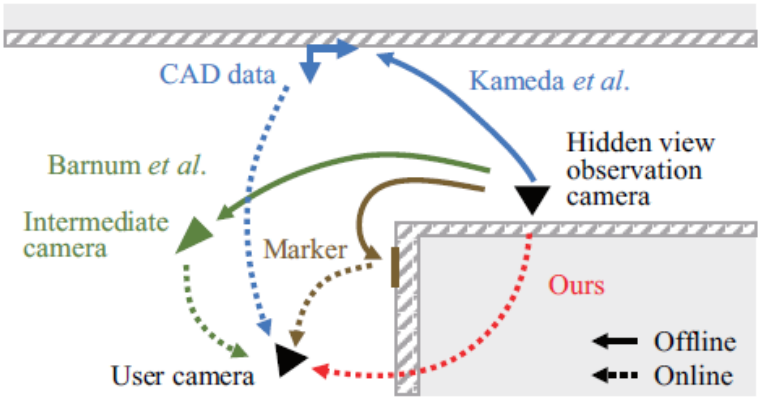


Figura 10 – Confronto tra i tipici metodi e quello proposto, articolo AISTVSUAWFOVCA3L

La figura successiva rappresenta invece il setup sperimentale per la cattura della scena tridimensionale:

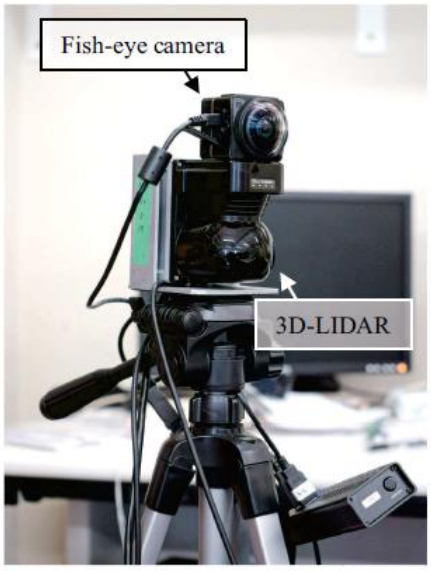


Figura 11 – RGB-D camera fish-eye e laser 3D, articolo AISTVSUAWFOVCA3L

2.4.3 – The Proposed Framework

La posizione della camera utente è calcolata sulla base della posizione 3D di alcuni punti fiduciali. Naturalmente, la differenza tra le immagini acquisite da una videocamera ad occhio di pesce e le immagini acquisite dalla camera pinhole, sono eccessive per poter confrontare direttamente detti punti caratteristici e, per questa ragione, occorre anzitutto convertire le immagini della camera ad occhio di pesce in immagini stile pinhole. Dopo di che, è possibile ottenere la "visione attraverso" sovrapponendo alla vista dell'utente un'immagine generata da nuvole di punti e l'immagine della camera ad occhio di pesce.

2.4.4 – User Camera Pose Estimation

Una camera ad occhio di pesce ha idealmente una proiezione equidistante. Tuttavia, le camere ad occhio di pesce attuali, non sono sempre ben rappresentate dal modello di proiezione ideale. Per questa ragione, si sono dapprima stimati i parametri interni della camera ad occhio di pesce utilizzando un modello proposto da Scaramuzza e la sua équipe. Si sono calcolati così i raggi corrispondenti ad ogni pixel dell'immagine ad occhio di pesce. Per renderizzare l'immagine virtuale della camera C_v, la camera è stata settata all'origine della camera ad occhio di pesce C_f. Dati i raggi di ogni pixel della camera ad occhio di pesce [*f* sta per *fish*]:

$$\boldsymbol{p}_f = \left(X_f, Y_f, Z_f\right)^T$$

ogni pixel della camera virtuale $\boldsymbol{p}_v$ è calcolata come:

$$\boldsymbol{p}_v \cong \boldsymbol{ARp}_f$$

Dove **A** ed **R** sono matrici 3x3 rispettivamente associate a parametri intrinseci e di rotazione della camera virtuale.
É stata stimata la posizione della camera utente risolvendo il problema di percezione degli *n* punti (problema PnP) nota la posizione dei punti 3D corrispondenti tra camera virtuale e camera utente. A questo punto, viene utilizzato *RANSAC* per rimuovere valori anomali ed ottenere una corrispondenza attendibile. La *Figura 12* mostra un esempio del risultato del confronto tra alcuni punti caratteristici di ciascuna immagine. In particolare, è illustrato:

- In alto a sinistra, l'input della camera pinhole [rappresentativa dell'utente nascosto]
- Nell'immagine di destra, l'input della camera fisheye
- Nell'immagine in basso a sinistra, i risultati della visione attraverso la parete
- Le linee rappresentative delle corrispondenze trovate (linee rette, di vari colori)

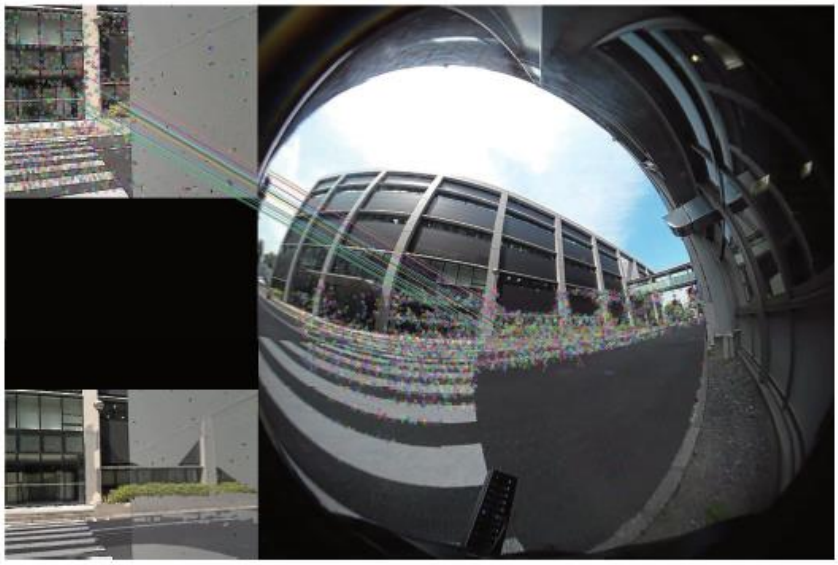


Figura 12 – Confronto punti rilevati da camera pinhole e fisheye, articolo AISTVSUAWFOVCA3L

2.4.5 – See-through Image Generation

La numerosità della nuvola di punti rilevata dal sistema laser 3D è troppo bassa per riempire la regione di interesse della vista dell'utente. Per questa ragione, è stato generata una mesh triangolare della nuvola di punti stessa mediante i pixel dell'immagine della camera ad occhio di pesce.

2.4.6 – Performance Validation

É stata posizionata una telecamera [rappresentativa dell'osservatore nascosto] dietro una struttura, a circa 100 cm di altezza, come mostrato in *Figura 13,* e catturata una porzione di muro con detta camera. A titolo di esempio, è stata scelta una scena in cui una persona sta camminando d'innanzi alla struttura. In questo esperimento, il processo temporalmente maggiormente dispendioso è costituito dalla stima della posizione della camera rappresentativa dell'utente nascosto.

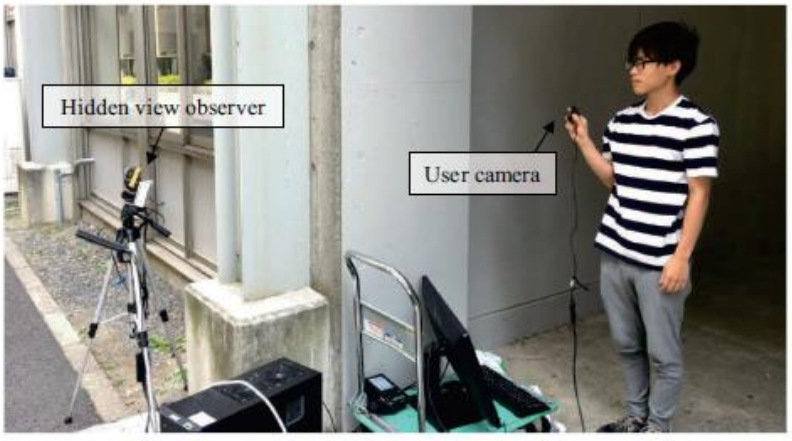


Figure 4: Experimental setup

Figura 13 – Setup sperimentale, articolo AISTVSUAWFOVCA3L

2.4.7 – Robustness

Si è ipotizzato che il tasso di successo del sistema di visione attraverso decade con la distanza dall'oservatore di vista occultata. Perciò, si è sistematicamente cambiata la posizione relativa della camera utente e misurato il tasso di successo di ogni punto. Il tasso di successo è stato definito come rapporto tra il numero di fotogrammi in cui lo sfondo è stato ricostruito correttamente con 10 secondi di "visione attraverso". La mostra i risultati dell'esperimento:

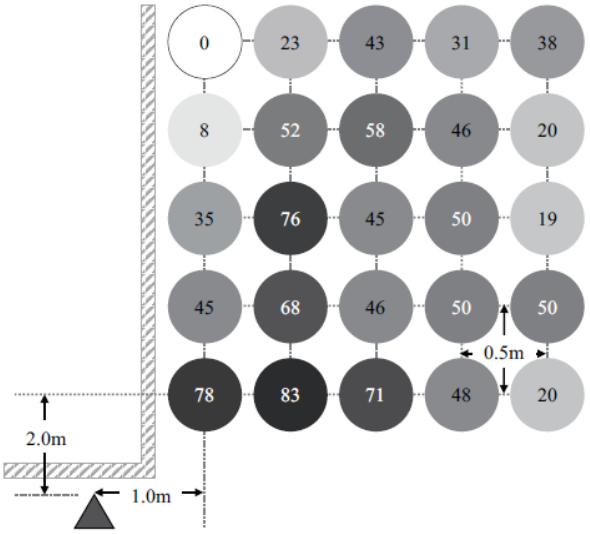


Figura 14 – Misura del tasso di successo del sistema proposto, articolo AISTVSUAWFOVCA3L

2.4.8 – Applicability

Di seguito, le immagini dell'esperimento effettuato in altri ambienti. Rispetto agli altri metodi, il nostro è molto portabile in quanto non richiede la pre calibrazione dello sfondo della camera dell'osservatore. In *Figura 15*, i risultati del sistema proposto. Si noti che la persona che sta camminando dietro al muro è indicativa del fatto che la scena è tempovariante. Più in basso, è riportato uno zoom sulla zona ritenuta di particolare interesse.

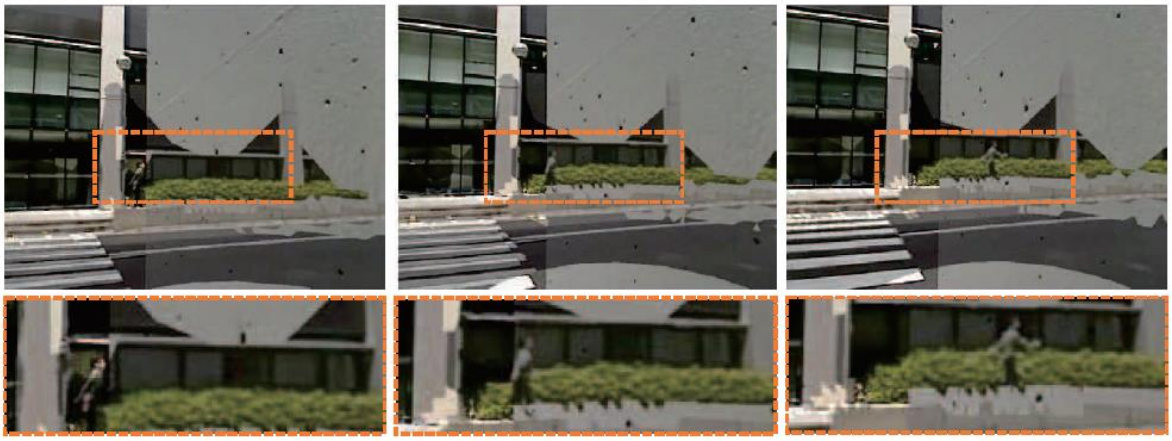


Figura 15 – Risultati del sistema di visione attraverso proposto, articolo AISTVSUAWFOVCA3L

2.5 – Development of a detection road users system in vehicle A-pillar blind spots

2.5.1 – Abstract

Questo articolo presenta un sistema che consente di espandere la visibilità dell'automobile utilizzando dei display nei fissati sui montanti del parabrezza. Non essendo il solo uso dei display particolarmente sicuro, il sistema viene correlato dal rilevamento di pedoni e oggetti in probabile collisione. Il tutto consta di modulo computer a bordo, una videocamera ed un display. Gli specchietti esterni sono stati rimpiazzati da videocamere.

Attualmente, vi sono numerose soluzioni disponibili su Mercedes, BMW, Volvo, Tesla, Volkswagen, Toyota, Nissan, Ford, ecc proposte come "*Control system of blind spots*". Tutti gli algoritmi d'azione sono però simili. Nel caso di Jaguar, il sistema "transparent pillars" fa si che vicino agli incroci o durante il cambio di traiettoria i montanti divengano automaticamente "invisibili".

2.5.2 – Development of the system

Come da *Figura 16*, il sistema consiste di una videocamera che cattura il video dall'esterno, un unità di calcolo basata su quella per lo sviluppo della Nvidia Jetson TX2, ed un display. La telecamera della Logitech è utilizzata per problemi di rilevamento oggetti in zona nascosta.

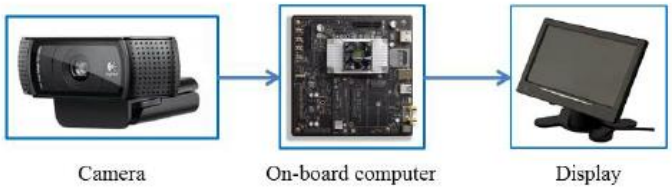


Figure 4. System interaction diagram

Figura 16 – Diagramma d'interazione del sistema, articolo DOADRUSIVA-PBS

In *Figura 17* la fotografia del modulo "computer di bordo":

Figure 5. Assembling the on-board computer module

Figura 17 – Assemblaggio del modulo computer di bordo, articolo DOADRUSIVA-PBS

Il modulo colleziona informazioni dalle camere (montate al posto degli specchietti retrovisori) ed è connesso al Can-bus dell'auto per raccogliere dati relativi alla velocità del veicolo. Secondo questa filosofia, si è così sviluppato l'interno visibile in *Figura 18*:

Figura 18 – Interni del GAZelle NEXT, articolo DOADRUSIVA-PBS

Come visibile in *Figura 19*, è stato inoltre pensato di utilizzare delle videocamere anche per la visione posteriore.

Figura 19 – Specchietti per il fissaggio della videocamere, articolo DOADRUSIVA-PBS

2.5.3 – System tests

Come anticipato, sono stati effettuati dei test con l'aggiunta di un sistema di controllo della collisione con i pedoni. Il sistema determina la velocità del movimento e rileva l'oggetto con l'aiuto della video camera, e ne considera la distanza con esso.

Figura 20 – Arresto con successo di fronte ad un pedone fantoccio, articolo DOADRUSIVA-PBS

In *Figura 21*, un'illustrazione del sistema in condizioni di esercizio prive di allerta possibili collisioni:

Figura 21 – Sistema operante senza l'allerta di possibili collisioni, articolo DOADRUSIVA-PBS

2.2.4 – Conclusioni

Il sistema installato ha consentito di mantenere una visibilità adeguata alla maggior parte delle circostanze, nonchè di aumentare l'ergonomia dell'interno. Il rilevamento dei pedoni si è mostrato relativamente stabile, ma occorre migliorarlo. In particolare, si rende al solito necessario effettuare il debug dei coefficienti tenendo in considerazione il tempo di approccio agli oggetti. Inoltre, i test sono stati fatti su una strada coperta di neve. Questo porta al fatto che lo spazio di frenata necessita di più del doppio che nel caso di strada asciutta. Tale fenomeno richiede accurati studi per minimizzare l'influenza di questo fattore sulla corretta operatività del sistema.

3 – INTRODUZIONE AL PINHOLE CAMERA MODEL

3.1 – Imaging Geometry

Come anticipato all'interno dell'Abstract, **il presente capitolo ed i successivi saranno sviluppati per semplicità *sostituendo agli occhi del guidatore una "camera ad esso equivalente"***.
In definitiva, si auspica quindi ad arrivare a confrontare l'immagine acquisita sperimentalmente dalla fotocamera posta a fare le veci del pilota (rinominata con C_0), con l'immagine sintetizzata utilizzando fotogrammi acquisiti da camere fissate d'innanzi al montante A.

Dopo aver letto con attenzione gli articoli proposti nel *Capitolo 2*, si comprende a maggior ragione la necessità di disporre di un modello matematico descrittivo delle fotocamere utilizzate.

Si utilizzerà per semplicità il modello matematico che va sotto il nome di *Pinhole Camera Model* (anche detto *Imaging geometry*, *Perspective Transformation*, *Projective Geometry, Image Formation Process, Central Projection Model, Imaging Transformation*). Non verranno pertanto contemplate possibili distorsioni delle lenti, e si avrà anzitutto a che fare con un semplice modello costituito da equazioni algebriche lineari.

Si è anzitutto risolto il problema dal punto di vista ideale. In questa sede si è infatti trascurato il fatto che un punto P esistente nel mondo tridimensionale e descritto dalle componenti del vettore $\vec{w}$, se proiettato sul piano immagine della camera non sarà perfettamente rappresentato in virtù del fatto che i pixel sono dotati di una certa estensione. Per questa ragione, il modello matematico di riferimento è quello rappresentato in *Figura 22*:

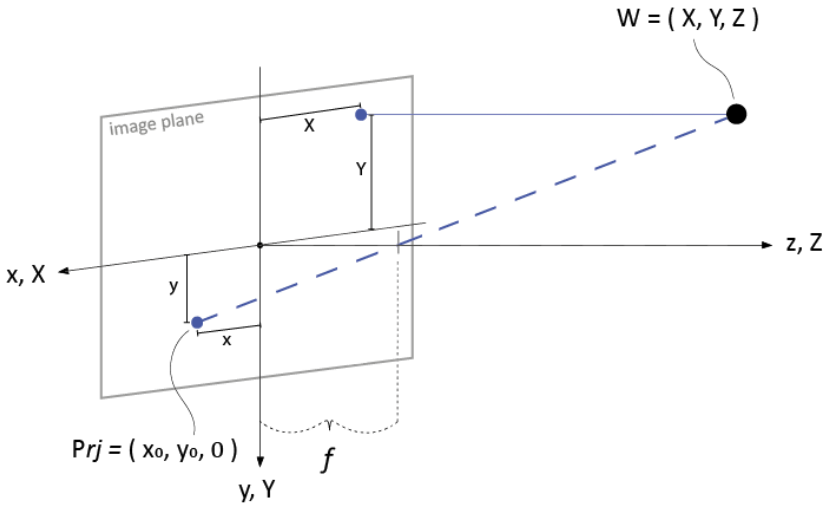


Figura 22 – Modello matematico della Camera

Con riferimento allo schema illustrato pocanzi, la semplice proporzionalità dei triangoli impone che le coordinate della proiezione di un punto P sul piano immagine della camera siano:

$$x = f\frac{X}{f - Z} \quad ; \quad y = f\frac{Y}{f - Z} \tag{1}$$

Per comodità di manipolazione matematica, tale espressione può essere anche rappresentata in forma matriciale, più nello specifico facendo ricorso alle coordinate omogenee.

In generale, un punto P espresso dalle tre componenti cartesiane rispetto alla base principale ("*w*" di "*world coordinates*") risulta:

$$\vec{w} = [X\ Y\ Z]^T$$

Il medesimo punto P espresso in coordinate omogenee diviene invece:

$$\overrightarrow{w_h} = \begin{bmatrix} k\,X \\ k\,Y \\ k\,Z \\ k \end{bmatrix} \quad ; \quad k \neq 0$$

E' possibile effettuare anche il processo inverso: per passare da un punto P espresso in coordinate omogenee $\overrightarrow{w_h}$, al punto P espresso in coordinate cortesiane $\vec{w}$, è evidentemente sufficiente dividere per lo scalare k e dimenticarsi dell'ultima componete. In simboli:

$$\overrightarrow{w_h} = \begin{bmatrix} k\,X \\ k\,Y \\ k\,Z \\ k \end{bmatrix} \rightarrow \begin{bmatrix} k\,X/k \\ k\,Y/k \\ k\,Z/k \\ k/k \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \vec{w} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Sotto questo scenario è facile verificare che il sistema matriciale corrispondente alla (1) ed espresso in coordinate omogenee è il seguente:

$$\overrightarrow{C_h} = \underline{P}\ \overrightarrow{w_h} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/f & 0 \end{bmatrix} \begin{bmatrix} k\,X \\ k\,Y \\ k\,Z \\ k \end{bmatrix} = \begin{bmatrix} k\,X \\ k\,Y \\ k\,Z \\ -k\dfrac{Z}{f} + k \end{bmatrix}$$

Ora, per abbandonare le coordinate omogenee e tornare alle coordinate cartesiane è sufficiente, al solito, dividere per l'ultima componente del vettore appena trovato, ottenendo quindi:

$$\overrightarrow{C_h} = \begin{bmatrix} k\,X \\ k\,Y \\ k\,Z \\ -k\dfrac{Z}{f} + k \end{bmatrix} \rightarrow \begin{bmatrix} k\,X/(-k\dfrac{Z}{f} + k) \\ k\,Y/(-k\dfrac{Z}{f} + k) \\ k\,Z/(-k\dfrac{Z}{f} + k) \\ -k\dfrac{Z}{f} + k/(-k\dfrac{Z}{f} + k) \end{bmatrix} = \begin{bmatrix} f\,X/(f-Z) \\ f\,Y/(f-Z) \\ f\,Z/(f-Z) \\ 1 \end{bmatrix} \rightarrow \vec{C} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f\dfrac{X}{f-Z} \\ f\dfrac{Y}{f-Z} \\ f\dfrac{Z}{f-Z} \end{bmatrix}$$

Si noti che le prime due componenti del vettore $\vec{C}$ sono proprio l'equazione 1.
$\underline{P}$ prende il nome di matrice di Percezione.

3.2 – Generalizzazione del modello

Si vogliono ora determinare le coordinate di un punto proiettato sul piano immagine di una qualunque camera. In altre parole, si contempla la possibilità che il piano immagine della fotocamera non sia necessariamente coincidente con la posizione ed orientamento della terna principale (detta "*world*"). Per fare ciò, è utile ancora una volta appoggiarsi alle trasformazioni omogenee, poiché consentono il passaggio da un sistema di riferimento ad un altro per mezzo di semplici moltiplicazioni matriciali.
Ci si accontenterà in questa sede di citare esempi per l'ottenimento delle coordinate in forma numerica, e non a dimostrare l'espressione analitica completa delle stesse.

Supponendo note le coordinate di un punto P rispetto alla terna principale (X,Y,Z;O), si vogliono ottenere le coordinate della proiezione di P sul piano immagine di una fotocamera arbitrariamente orientata e posizionata. Sia tale punto denominato *Prj*. Si stanno in altre parole ricercando le coordinate di *Prj* sia rispetto al piano immagine (x,y,z;C_1), sia secondo la terna principale (X,Y,Z;O).

A titolo di esempio, si supponga che la camera osservante (C_1) sia caratterizzate dalle seguenti coordinate cartesiane rispetto alla base principale O:

$$\overrightarrow{d_O^{C1}}_O = [X_{C1} \quad Y_{C1} \quad Z_{C1}]_O = [-.31 \quad 1.02 \quad .155] \quad (m)$$

(Leggasi "*d da C1 ad O, espresso in base O*")

E' immediato verificare che, in assenza di sfasamento relativo tra gli assi della terna principale O e quelli della terna della camera C_1, le relazioni tra un medesimo punto P visto da entrambi gli osservatori sono espressi dalle seguenti:

$$\begin{cases} x = X - \left. d_O^{C1} \right|_X \\ y = Y - \left. d_O^{C1} \right|_Y \\ z = Z - \left. d_O^{C1} \right|_Z \end{cases}$$

Equivalentemente, è possibile esprimere detto sistema mediante il formalismo matriciale:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -\left. d_O^{C1} \right|_X \\ 0 & 1 & 0 & -\left. d_O^{C1} \right|_Y \\ 0 & 0 & 1 & -\left. d_O^{C1} \right|_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

In questo modo, chiamata $\underline{Tr}\,_O^{C1}$ la matrice di traslazione omogenea, per passare dal sistema (X,Y,Z;O) al sistema (x,y,z;C1) è possibile effettuare il semplice prodotto matriciale:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \underline{Tr}\,_O^{C1} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

La moltiplicazione per $\underline{Tr}\,{}_{O}^{C1}$ corrisponde dunque al passaggio dalle coordinate in base O alle coordinate in base C_1. Alla matrice di traslazione omogenea $\underline{Tr}\,{}_{O}^{C1}$ corrisponde graficamente la situazione illustrata in Figura 23:

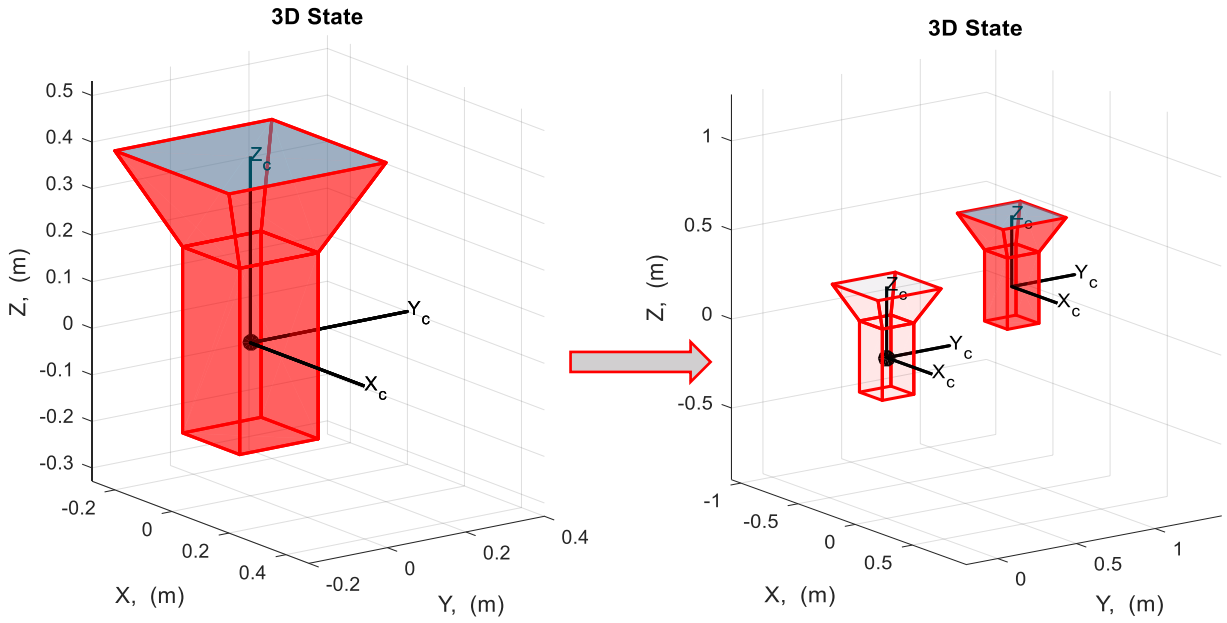


Figura 23 – Esempio applicativo trasformazioni omogenee: matrice di traslazione

Secondo la stessa logica, è possibile ridefinire l'orientamento del nuovo sistema di riferimento, ovvero della camera C_1. Ci si avvale ancora delle trasformazioni omogenee, dove tuttavia in questo caso la matrice di trasformazione sarà composta da matrici di rotazione.
Si supponga a tal proposito di voler orientare l'asse ottico della camera (z) nella direzione Y positivo. Occore evidentemente effettuare una rotazione attorno all'asse x della camera, denominato da Matlab "X_c". Detto α l'angolo di 90° di cui ruotare la telecamera, la prima matrice di rotazione 3X3 (positiva) assume la seguente forma:

$$\underline{FirstR}_{C1}^{O} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

La trasformazione ad essa corrispondente è esprimibile mediante le corrispettiva matrice omogenea 4x4. Poichè tuttavia la rotazione andrebbe interpretata *come l'angolo di cui ruotare il nuovo sistema ricercato perchè coincida con il vecchio*, la rotazione deve essere vista "al rovescio". Ciò corrisponde matematicamente all'utilizzo della matrice inversa di quella appena descritta. In simboli:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Come aggiunta al movimento precedente, essa comporta la situazione di *Figura 24*:

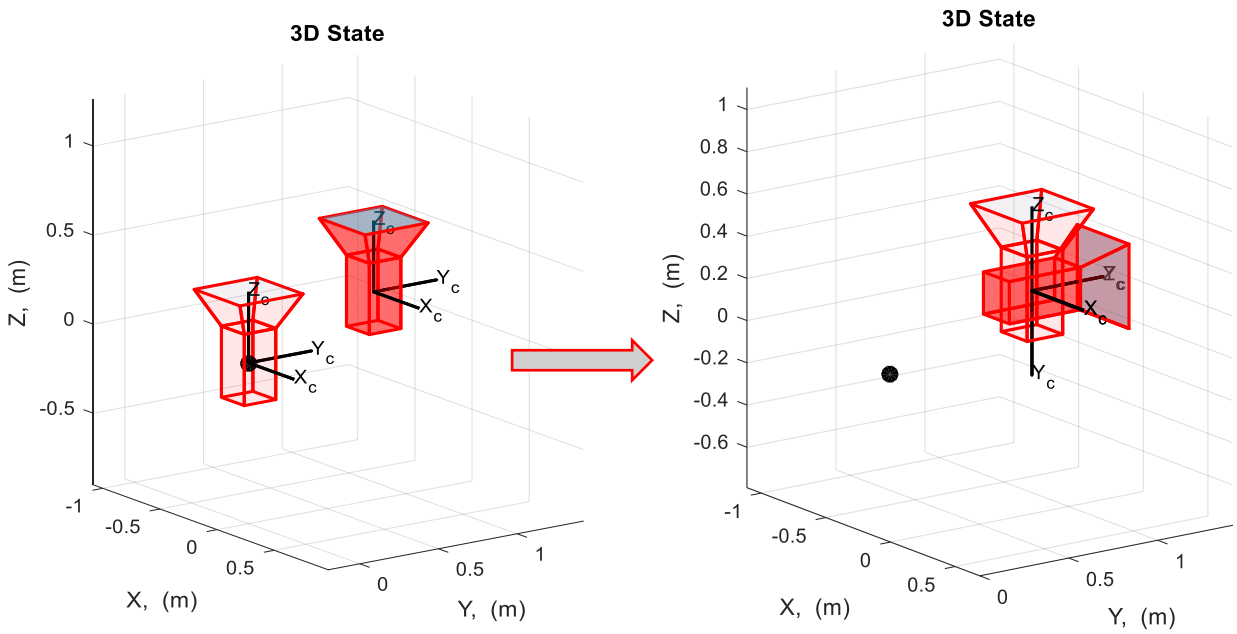


Figura 24 – Esempio applicativo trasformazioni omogenee: prima matrice di rotazione

Si ipotizzi infine che la camera sia ruotata di 40° positivi attorno all'attuale asse y, denominato da Matlab "Y_c". Trattasi di una rotazione piana attorno al secondo asse, pertanto esprimibile con la seguente matrice di rotazione:

$$\underline{SecondR}_{C1}^{O} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

Analogamente al caso precedente, la trasformazione ad essa corrispondente è esprimibile mediante le corrispettiva matrice 4x4:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Essa comporta ora la situazione tridimensionale di *Figura 25*:

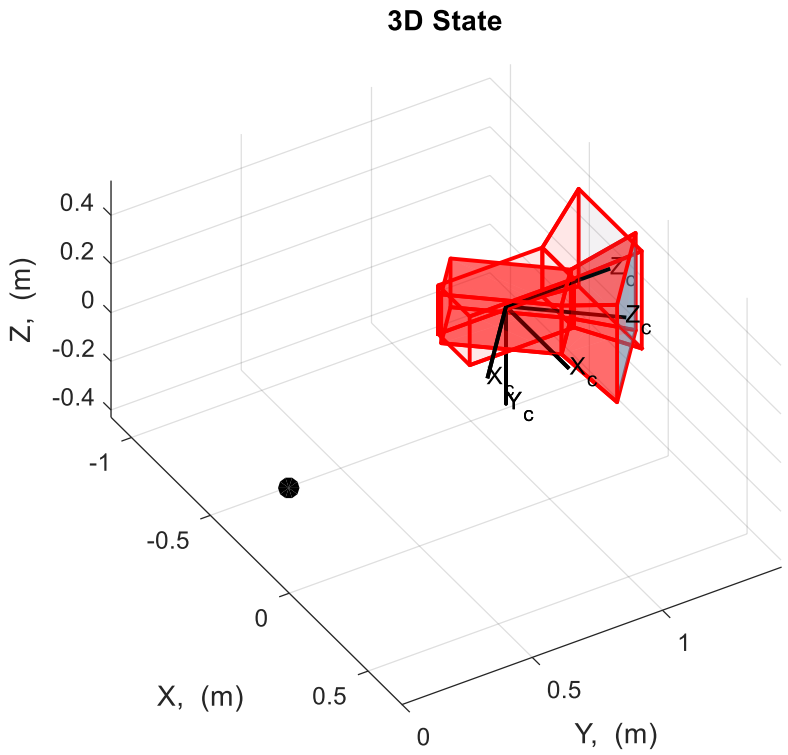


Figura 25 – Esempio applicativo trasformazioni omogenee: seconda matrice di rotazione

In definitiva si è dunque complessivamente operata una trasformazione omogenea matematicamente descritta dai seguenti prodotti in cascata:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & -d_O^{C1}]_X \\ 0 & 1 & 0 & -d_O^{C1}]_Y \\ 0 & 0 & 1 & -d_O^{C1}]_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Per compattezza, è possibile rinominare con $\underline{H}\,{}_O^{C1}$ la matrice di trasformazione omogenea composta dalle tre matrici connesse ai movimenti sovra descritti. In simboli:

$$\underline{H}\,{}_O^{C1} = \begin{bmatrix} (SecondR\,{}_{C1}^{O})^{-1} & \vec{0} \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} (FirstR\,{}_{C1}^{O})^{-1} & \vec{0} \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} \underline{Tr}\,{}_O^{C1} & \vec{0} \\ \vec{0} & 1 \end{bmatrix}$$

Si ha dunque che il cambio di base è completamente descritto dalla seguente espressione:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \underline{H}\,{}_O^{C1} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Ci si è ora ricondotti al caso del modello semplice: sono note le coordinate di un generico punto P rispetto al sistema di riferimento della camera $[x \ y \ z \ 1]$, e si vuole determinare le coordinate della proiezione *Prj* di detto punto sul piano immagine. Per fare ciò, si utilizzerà l'espressione (1) e si utilizzerà in seguito il modello matematico della *Pinhole Camera*.

Si concepisce così uno script Matlab per applicare i concetti appena appresi da un punto di vista più pratico. Si decide anzitutto di valutare come una lettera "**F**" fissata sul piano ZX sia proiettata su differenti piani immagine associati a tre fotocamere svariatamente orietate. I risultati della semplice proiezione di ciascun punto della lettera **F** sono mostrati in *Figura 26*:

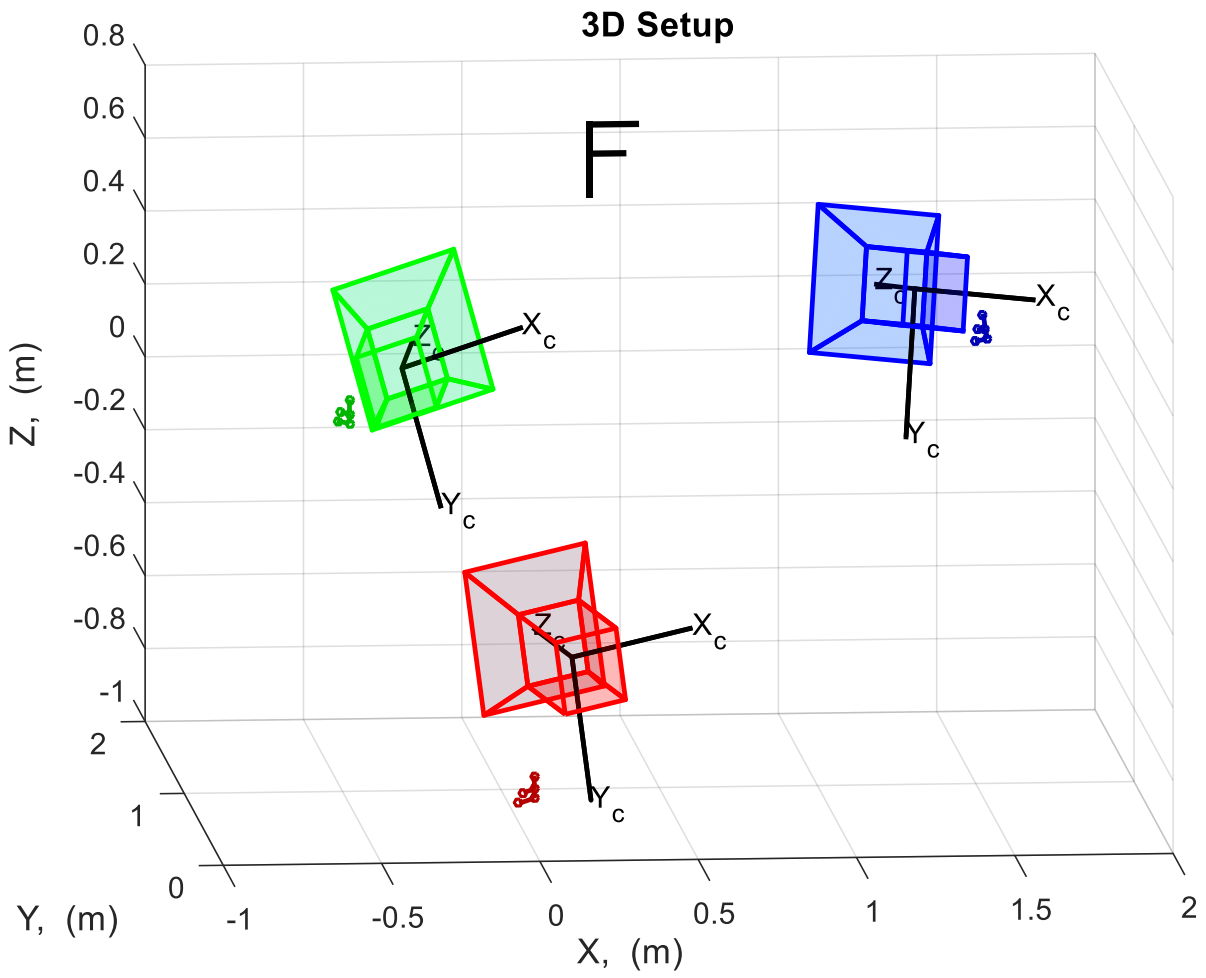


Figura 26 – Riproiezioni lettera F su differenti piani immagini

Come test successivo, si sceglie di eseguire un test sperimentale. Si posiziona ed orienta una fotocamera in modo più attinente possibile ai parametri di input del modello di proiezione appena esposto. In questa sede, si mira quindi a valutare da un punto di vista macroscopico (senza grosse pretese dunque) la coerenza del modello appena concepito. Il setup sperimentale è mostrato in Figura 27:

Figura 27 – Setup sperimentale per test proiezione

Assumendo che la lunghezza focale della fotocamera C_1 sia di circa 0.15 m, l'esito del modello matematico messo a confronto con l'immagine acquisita dalla fotocamera (si è utilizzata la fotocamera di un *Samsung Galaxy S6 Edge*) è mostrato in *Figura 28*:

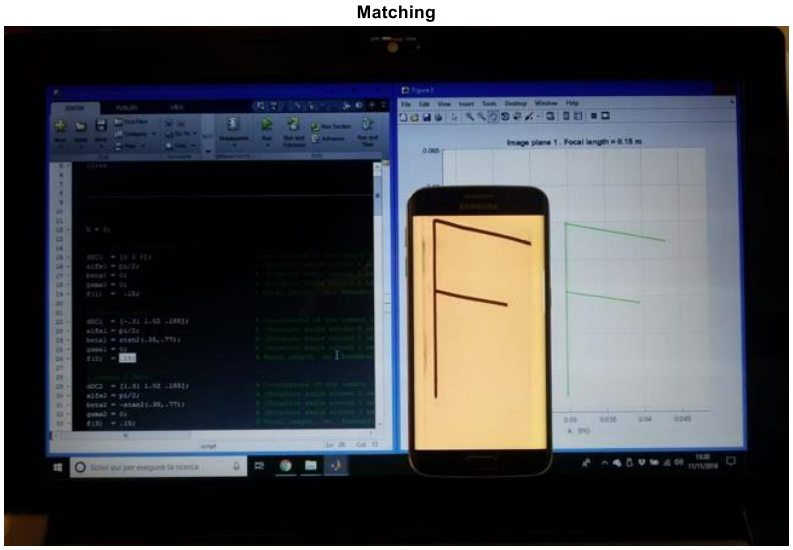


Figura 28 – Test proiezione VS foto acquisita da fotocamera Samsung

3.3 – Inverse Perspective Geometry

3.3.1 – Perché l'Inverse Perspective Geometry

Presa la sufficiente confidenza con le trasformazioni omogenee e con il modello matematico della fotocamera, si considera ora il medesimo problema ma dotato di un differente flusso logico, più simile a quello che poi sarà il problema riscontrato sul campo dal vero e proprio *Sistema di identificazione scena stradale*.

In questa sede si vuole ricostruire la lettera **F** sul piano immagine dell'osservatore C_0, note le coordinate dei corrispondenti punti di **F** rilevati dalle camere C_1 e C_2. Dal punto di vista dei risultati, ci si aspetta quindi di ottenere i medesimi valori dei precedenti programmi di proiezione, mentre ciò che cambia, come anticipato, è il flusso logico di ottenimento dei risultati stessi.

Prima di esporre il diagramma a blocchi del programma "di ricostruzione della lettera **F**", occorre effettuare un richiamo teorico. Nello specifico, si pone ora l'attenzione sul fatto che il problema *Inverse Perspective Geometry* sia leggermente più complesso del corrispettivo *Direct Perspective Geometry*. Questo è conseguenza del fatto che proiettando dei punti 3D sulla superficie di un piano a distanza nota, il processo è univoco. Di contro, se sulla base dei punti appartenenti al piano si cerca di comprendere quali siano i punti tridimensionali "genitori", il processo non è più univoco. A priori infatti, tutti i punti 3D appartenenti alla retta di proiezione sul piano, soddisfano il fatto di andare a ricadere sulla superficie stessa del piano. Questa asserzione è illustrata schematicamente in *Figura 29*:

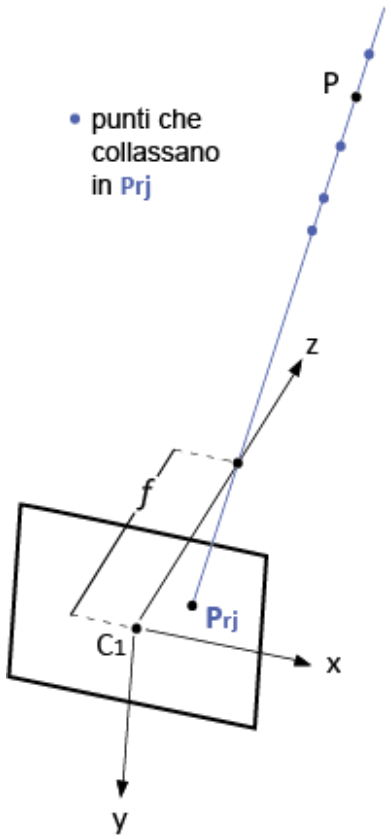


Figura 29 – Introduzione al problema Inverse Perspective Geometry

Per questa ragione, si può pensare di ottenere l'informazine aggiuntiva sulla profondità mediante l'uso di un ulteriore fotocamera, proprio come avviene nel sistema visivo umano.
Collocando infatti una camera aggiuntiva, le triangolazioni impongono che il problema sia nuovamente univocamente definito. A tal proposito, si faccia riferimento alla *Figura 30*:

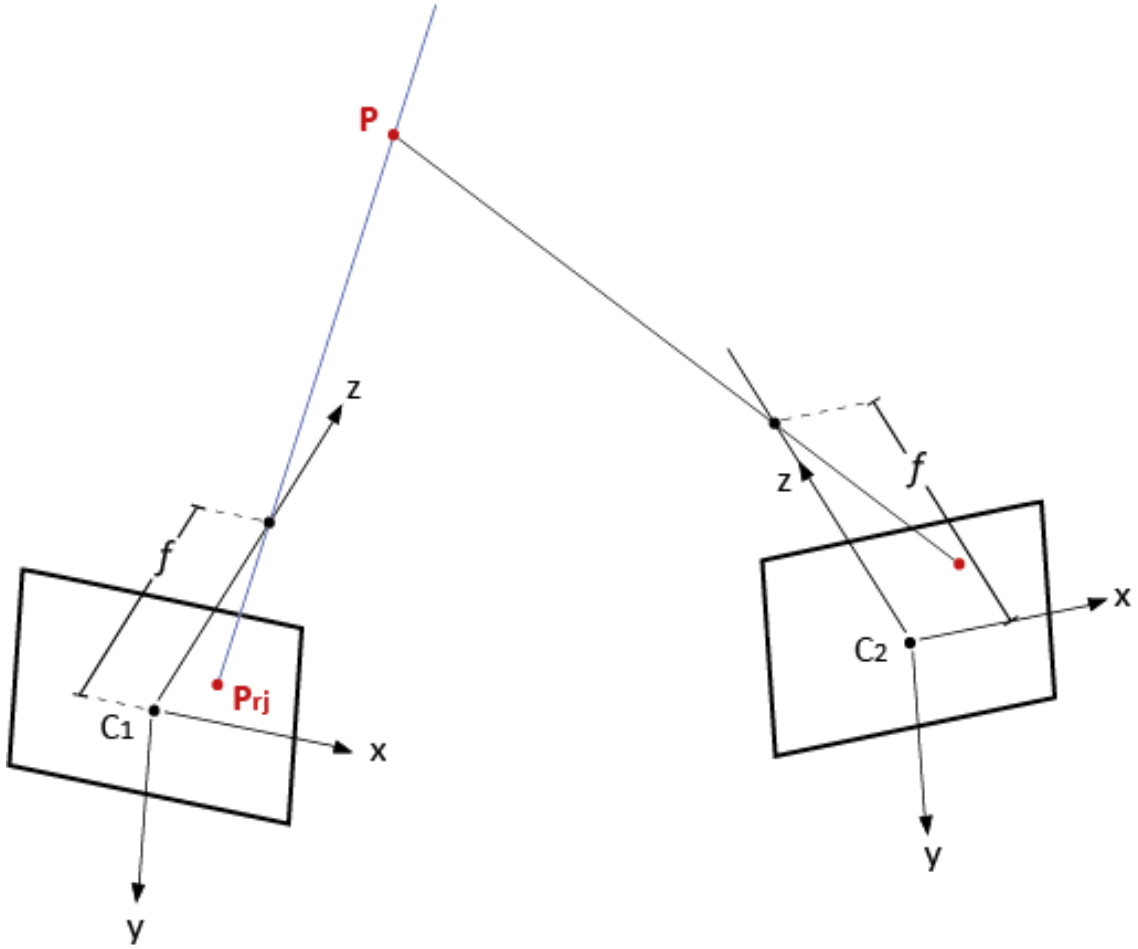


Figura 30 – Schema Inverse Perspective Geometry

3.3.2 – Logica di programmazione

Sotto questo prospetto, ci si accinge ora al concepimento del flusso logico per la deduzione della posizione della lettera **F** sulla base di coordinate immagine in arrivo dalle sole due camere C_1 e C_2.

Siano:

- Matrix_F, la matrice nx3 che raccoglie le coordinate dei punti che formano la lettera F, con n = numero di punti (evidentemente, per la lettera "F" si ha che n = 5)
- PrjBaseC1, la matrice omogenea nx4 che raccoglie le proiezioni dei punti Matrix_F sul piano immagine della camera C_1
- PrjBaseC2, la matrice omogenea nx4 che raccoglie le proiezioni dei punti Matrix_F sul piano immagine della camera C_2
- PbaseC1, la matrice omogenea nx4 che raccoglie le riproiezioni dei punti PrjBaseC1 nel mondo 3D, rispetto alla base C_1
- PbaseO, la matrice omogenea nx4 che raccoglie le riproiezioni dei punti PrjBaseC1 nel mondo 3D, rispetto alla base principale O
- Cam_Coords, le coordinate delle riproiezioni rispetto al sistema di riferimento C_0
- Img_Coords, le coordinate dell'immagine ricostruita che dovrebbe percepire C_0

Allora, il programma di test assume il seguente aspetto:

Si crea la matrice Matrix_F, assegnandole un set valori (coordinate) noti.
Mediante il modello della fotocamera, si calcolano le coordinate delle proiezioni dei cinque punti che formano la lettera **F** *su ciascun piano delle tre camere*. Si è così generato l'input funzionale vero e proprio per il programma: sono note le immagini della lettera **F** sulle camere C_1 e C_2 (in altre parole, sono note le coordinate "PrjBaseC1" e "PrjBaseC2"), e si mira a "ricostruire" la **F** che vive nel mondo tridimensionale e che è stata "catturata" dalle due camere stesse.
Si auspica infine a proiettare i punti della **F** del mondo (catturati mediante le camere "ausiliarie" C_1 e C_2) sul piano immagine dell'osservatore nascosto C_0.
Per ottenere tutto ciò, si affronta anzitutto il problema *Inverse Perspective Geometry*. Risolvendo il quesito infatti, si è in grado di ottenere i punti della lettera **F** che vive nel mondo, rispetto al sistema di riferimento di una delle due camere ausiliarie: in questo caso si è scelto C_1.
Facendo ricorso alle trasformazioni omogenee, mediante semplici prodotti si riottengono le coordinate di questo set di punti, ma rispetto all'origine assoluto O ("PBaseO"). Questo set di punti, verrà riproiettato sulla **F** del mondo, per verificare il corretto avvenimento della "triangolazione per la deduzione del 3D".
Infine, si proietta sul piano di C_0, C_1 e C_2 i punti appena generati. Per fare questo, ci si appoggia ad una semplice *Direct Perspective Geometry*, preventivamente "aggiustata" mediante le classiche trasformazioni omogenee.
Convenzionalmente, si sceglie di rappresentare mediante dei Marker circolari i punti riproiettati.

La descrizione grafica dell'algoritmo appena discusso è rappresentata in *Figura 31*:

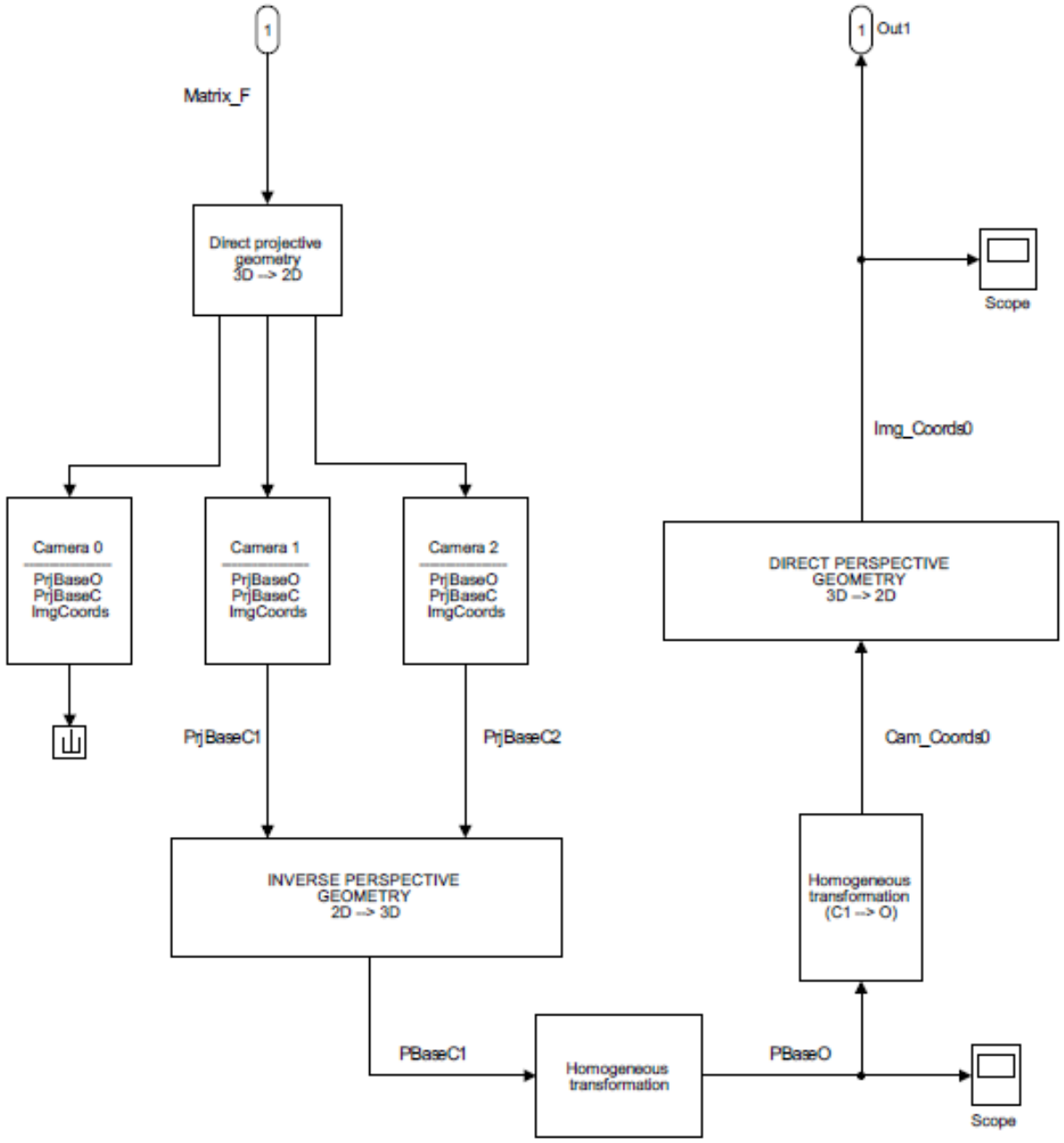


Figura 31 – Schema a blocchi del programma di ricostruzione della F

3.3.3 – Equazioni risolutive, caso piano

Data la complessità computazionale del calcolo delle proiezioni in caso di una mole massiccia di punti (tipicamente *n* diverrà molto grande), e la necessità di ottenere successivamente una generazione dell'immagine in Real-time, si opta per risolvere in forma chiusa il problema geometrico.

Per semplificare il problema ed iniziare ad ottenere confidenza con le matrici, si opta dapprima per risolvere la geometria ipotizzando che le 2 camere ausiliarie (C_1 e C_2) siano sfasate *solo* attorno all'asse verticale. La situazione grafica è illustrata in *Figura 32*:

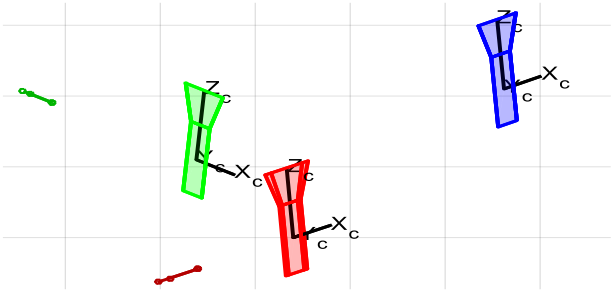


Figura 32 – Setup per algoritmo di riproiezione lettera F, caso piano

Detto β l'angolo compreso tra la direzione x_c della camera C_1 e la direzione x_c della camera C_2, facendo uso dei modelli di proiezione e delle trasformazioni omogenee presentate in precedenza, è facile convincersi che ci si pone di fronte al set di equazioni algebriche (2). Ogni riga è da interpretarsi come un vettore di *n* elementi associati agli *n* punti dell'immagine, in questo caso 5 (lettera **F**).

La simbologia utilizzata è descritta in *Tabella 2*:

NOTAZIONE UTILIZZATA NEL PROGRAMMA MATLAB	
SIMBOLO	DESCRIZIONE
$\boldsymbol{X_P\vert_1}$	Coordinate x del generico punto P nel mondo, rispetto al SDR di C_1
$\boldsymbol{Z_P\vert_1}$	Coordinate z del generico punto P nel mondo, rispetto al SDR di C_1
$\boldsymbol{Y_P\vert_1}$	Coordinate y del generico punto P nel mondo, rispetto al SDR di C_1
$\boldsymbol{X_P\vert_2}$	Coordinate x del generico punto P nel mondo, rispetto al SDR di C_2
$\boldsymbol{Z_P\vert_2}$	Coordinate z del generico punto P nel mondo, rispetto al SDR di C_2
$\boldsymbol{X_{C_2}\vert_1}$	Coordinata x del punto C_2 visto dal SDR di C_1
$\boldsymbol{Z_{C_2}\vert_1}$	Coordinata z del punto C_2 visto dal SDR di C_1
f_1	Lunghezza focale della camera ausiliaria C_1
f_2	Lunghezza focale della camera ausiliaria C_2
$\boldsymbol{Prj_2(1)}$	Prime componenti delle proiezioni dei punti del mondo sul piano immagine della camera C_2
$\boldsymbol{Prj_1(1)}$	Prime componenti delle proiezioni dei punti del mondo sul piano immagine della camera C_1
$\boldsymbol{Prj_1(2)}$	Seconde componenti delle proiezioni dei punti del mondo sul piano immagine della camera C_1

Tabella 2 – Notazione utilizzata nel programma Matlab

Si avrà quindi:

$$\begin{cases} \boldsymbol{X_P}|_{\boldsymbol{1}} = \dfrac{\boldsymbol{Prj_1}(:,\boldsymbol{1})}{f_1} .* (f_1 - \boldsymbol{Z_P}|_{\boldsymbol{1}}) \\ \boldsymbol{Y_P}|_{\boldsymbol{1}} = \dfrac{\boldsymbol{Prj_1}(:,\boldsymbol{2})}{f_1} .* (f_1 - \boldsymbol{Z_P}|_{\boldsymbol{1}}) \\ \boldsymbol{X_P}|_{\boldsymbol{2}} = \dfrac{\boldsymbol{Prj_2}(:,\boldsymbol{1})}{f_1} .* (f_2 - \boldsymbol{Z_P}|_{\boldsymbol{2}}) \\ \boldsymbol{X_P}|_{\boldsymbol{1}} = \boldsymbol{X_P}|_{\boldsymbol{2}} \cos(\beta) + \boldsymbol{Z_P}|_{\boldsymbol{2}} \sin(\beta) + \boldsymbol{X_{C_2}}|_{\boldsymbol{1}} \\ \boldsymbol{Z_P}|_{\boldsymbol{1}} = -\boldsymbol{X_P}|_{\boldsymbol{2}} \sin(\beta) + \boldsymbol{Z_P}|_{\boldsymbol{2}} \cos(\beta) + \boldsymbol{Z_{C_2}}|_{\boldsymbol{1}} \end{cases} \quad (2)$$

Si noti che l'operatore "$.*$" sta a rimarcare che si tratta della moltiplicazione *elemento per elemento* tra entità vettoriali. Ciascun vettore è associato infatti alla lista degli *n* punti dell'immagine.

A titolo di esempio, per trovare le coordinate del *quarto* punto che forma la lettera **F** nel mondo, si dovrà utilizzare il sistema (2) compilato per il caso particolare di elemento 4. Pertanto, le incognite del sistema (3) sono solo 5 delle 5*n* totali:

$$\begin{cases} X_P|_1 = \dfrac{Prj_1(4,1)}{f_1} * (f_1 - Z_P|_1) \\ Y_P|_1 = \dfrac{Prj_1(4,2)}{f_1} * (f_1 - Z_P|_1) \\ X_P|_2 = \dfrac{Prj_2(4,1)}{f_1} * (f_2 - Z_P|_2) \\ X_P|_1 = X_P|_2 \cos(\beta) + Z_P|_2 \sin(\beta) + X_{C_2}|_1 \\ Z_P|_1 = -X_P|_2 \sin(\beta) + Z_P|_2 \cos(\beta) + Z_{C_2}|_1 \end{cases} \quad (3)$$

Dal punto di vista di utilità ingegneristica, di queste incognite si è interessati alle sole coordinate $X_P|_1$, $Y_P|_1$, $Z_P|_1$, ma la loro determinazione impone il coinvolgimento delle altre due equazioni. Più in generale, si dovrà quindi risolvere per ogni punto k degli *n* totali un sistema del tipo (3).

Dal punto di vista matematico, data la complessità nel risolvere il sistema per sostituzione, si sceglie di utilizzare il metodo sistematico di Cramer. Si riscrive quindi il sistema (3) in forma matriciale. Ponendo per brevità:

$$x_1 = X_P|_1$$

$$x_2 = Y_P|_1$$

$$x_3 = Z_P|_1$$

$$x_4 = X_P|_2$$

$$x_5 = Z_P|_2$$

$$k_1 = \frac{Prj_1(k,1)}{f_1}$$
$$k_2 = \frac{Prj_1(k,2)}{f_1}$$
$$k_3 = \frac{Prj_2(k,1)}{f_2}$$

$$k_4 = X_{C_2}|_1$$
$$k_5 = Z_{C_2}|_1$$

$$c = cos\ ;\ s = sin$$

Si è dinnanzi al classico sistema di equazioni lineare nelle incognite $\boldsymbol{x} = [x_1\ \ x_2\ \ x_3\ \ x_4\ \ x_5]^T$:

$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$$

Dove:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & k_1 & 0 & 0 \\ 0 & 1 & k_2 & 0 & 0 \\ 0 & 0 & 0 & 1 & k_3 \\ 1 & 0 & 0 & -c(\beta) & -s(\beta) \\ 0 & 0 & 1 & s(\beta) & -c(\beta) \end{bmatrix}$$

$$\boldsymbol{x} = [\ x_1\ \ x_2\ \ x_3\ \ x_4\ \ x_5\ ]^T = [\ X_P|_1\ \ Y_P|_1\ \ Z_P|_1\ \ X_P|_2\ \ Z_P|_2\ ]^T$$

$$\boldsymbol{b} = [\ k_1\, f_1\ \ k_2\, f_1\ \ k_3\, f_2\ \ k_4\ \ k_5\ ]^T$$

La soluzione è implementata all'interno del codice Matlab di peritnenza e da luogo alla *Figura 33*:

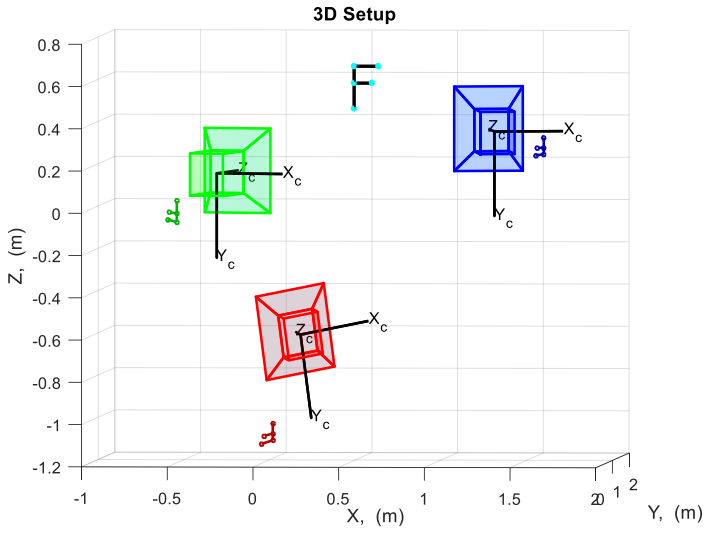


Figura 33 – Riproiezione lettera F, caso camere ausiliarie sfasate attorno ad un solo asse

3.3.4 – Equazioni risolutive, caso generale

La situazione, più generale, è illustrata in *Figura 34*:

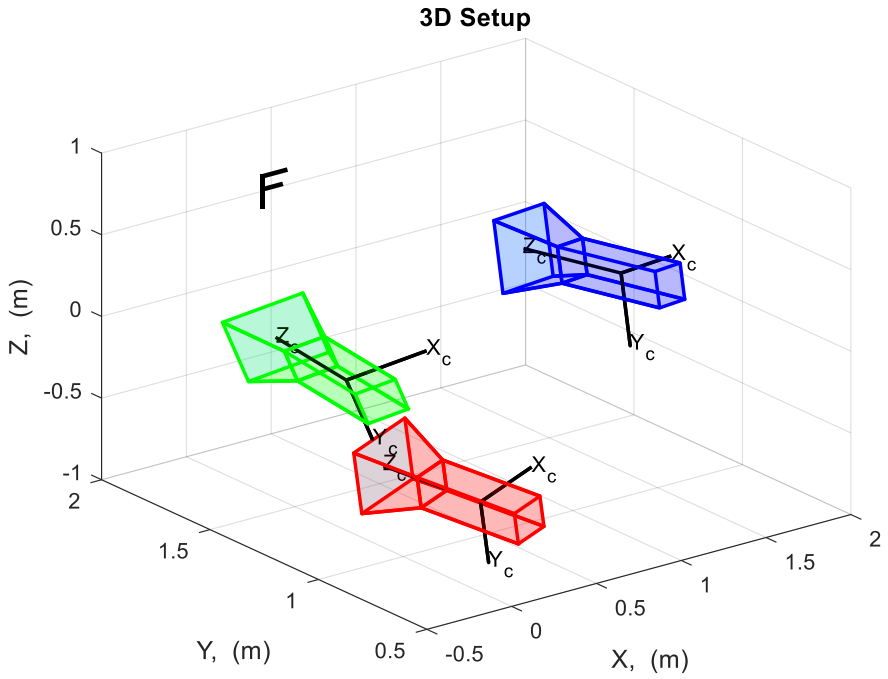


Figura 34 – Setup per algoritmo di riproiezione lettera F, camere arbitrariamente orientate

Si riportano di seguito le equazioni risolutive. Le tre equazioni di proiezione restano inalterate, ciò che cambia sono le espressioni per il passaggio tra i sistemi di riferimento delle due camere ausiliarie.

Supponendo di disporre degli angoli di rotazione tra le due camere C_1 e C_2, è possibile ottenere la matrice di rotazione complessiva tra i due sistemi svolgendo il seguente prodotto:

$$\boldsymbol{R} = \boldsymbol{R}_{\boldsymbol{z}}^{\boldsymbol{\alpha}}\, \boldsymbol{R}_{\boldsymbol{y}}^{\boldsymbol{\beta}}\, \boldsymbol{R}_{\boldsymbol{x}}^{\boldsymbol{\gamma}} = \begin{bmatrix} c(\alpha)c(\beta) & c(\alpha)s(\beta)s(\gamma) - s(\alpha)c(\gamma) & c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma) \\ s(\alpha)s(\beta) & s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma) & s(\alpha)s(\beta)c(\gamma) - c(\alpha)s(\gamma) \\ -s(\beta) & c(\beta)s(\gamma) & c(\beta)c(\gamma) \end{bmatrix}$$

Introducendo ora anche l'eventuale traslazione tra i due sistemi di riferimento ausiliari, la matrice di trasformazione omogenea $\boldsymbol{H}_{\boldsymbol{C_1}}^{\boldsymbol{C_2}}$ è composta dai seguenti elementi:

$$\begin{bmatrix} X_P|_1 \\ Y_P|_1 \\ Z_P|_1 \\ 1 \end{bmatrix} = \boldsymbol{H}_{\boldsymbol{C_1}}^{\boldsymbol{C_2}} \begin{bmatrix} X_P|_2 \\ Y_P|_2 \\ Z_P|_2 \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} c(\alpha)c(\beta) & c(\alpha)s(\beta)s(\gamma) - s(\alpha)c(\gamma) & c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma) & X_{C_2}|_1 \\ s(\alpha)s(\beta) & s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma) & s(\alpha)s(\beta)c(\gamma) - c(\alpha)s(\gamma) & Y_{C_2}|_1 \\ -s(\beta) & c(\beta)s(\gamma) & c(\beta)c(\gamma) & Z_{C_2}|_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_P|_2 \\ Y_P|_2 \\ Z_P|_2 \\ 1 \end{bmatrix}$$

Ovvero, genera le seguenti 3 equazioni:

$$\begin{cases} \boldsymbol{X_P}|_1 = \mathrm{c}(\alpha)\,\mathrm{c}(\beta)\ \boldsymbol{X_P}|_2 + \big(c(\alpha)s(\beta)s(\gamma) - s(\alpha)c(\gamma)\big)\,\boldsymbol{Y_P}|_2 + \big(c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma)\big)\,\boldsymbol{Z_P}|_2 + \ \boldsymbol{X_{C_2}}|_1 \\ \boldsymbol{Y_P}|_1 = s(\alpha)s(\beta)\,\boldsymbol{X_P}|_2 + \big(s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma)\big)\,\boldsymbol{Y_P}|_2 + \big(s(\alpha)s(\beta)c(\gamma) - c(\alpha)s(\gamma)\big)\,\boldsymbol{Z_P}|_2 + \ \boldsymbol{Y_{C_2}}|_1 \\ \boldsymbol{Z_P}|_1 = -\,\mathrm{s}(\beta)\,\boldsymbol{X_P}|_2 + c(\beta)s(\gamma)\boldsymbol{Y_P}|_2 + c(\beta)c(\gamma)\ \boldsymbol{Z_P}|_2 + \ \boldsymbol{Z_{C_2}}|_1 \end{cases}$$

Tenendo conto delle 3 equazioni di proiezione, e di queste ultime relazioni di cambio di base, si origina ancora una volta un sistema lineare, ora di dimensione 6x6:

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (4)$$

Dove la matrice di sistema assume la seguente forma:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & k_1 & 0 & 0 & 0 \\ 0 & 1 & k_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & k_3 & 0 \\ 1 & 0 & 0 & j_1 & j_2 & j_3 \\ 0 & 1 & 0 & j_4 & j_5 & j_6 \\ 0 & 0 & 1 & j_7 & j_8 & j_9 \end{bmatrix}$$

In cui:

$$\begin{array}{lll} j_1 = -\,\mathrm{c}(\alpha)\,\mathrm{c}(\beta) & ; \quad j_2 = -\big(c(\alpha)s(\beta)s(\gamma) - s(\alpha)c(\gamma)\big) & ; \quad j_3 = -\big(c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma)\big) \\ j_4 = -s(\alpha)s(\beta) & ; \quad j_5 = -\big(s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma)\big) & ; \quad j_6 = -\big(s(\alpha)s(\beta)c(\gamma) - c(\alpha)s(\gamma)\big) \\ j_7 = \mathrm{s}(\beta) & ; \quad j_8 = -c(\beta)s(\gamma) & ; \quad j_9 = -c(\beta)c(\gamma) \end{array}$$

In merito al vettore incognito ed al termine noto, si ha per definizione che:

$$\boldsymbol{x} = [\, x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5 \;\; x_6 \,]^T = [\, X_P|_1 \;\; Y_P|_1 \;\; Z_P|_1 \;\; X_P|_2 \;\; Z_P|_2 \;\; Y_P|_2 \;\;]^T$$

$$\boldsymbol{b} = [k_1\, f_1 \quad k_2\, f_1 \quad k_3\, f_2 \quad k_4 \quad k_6 \;\; k_5 \,]^T$$

Si riporta ora la soluzione di un caso generale. I punti evidenziati sono figli dello stesso flusso logico descritto nel paragrafo precedente.
Si noti che è possibile ottenere la soluzione in forma chiusa del sistema lineare (4) mediante l'uso del comando Matlab *syms*: ciò consente evidentemente un possibile successivo risparmio di tempo al computer.

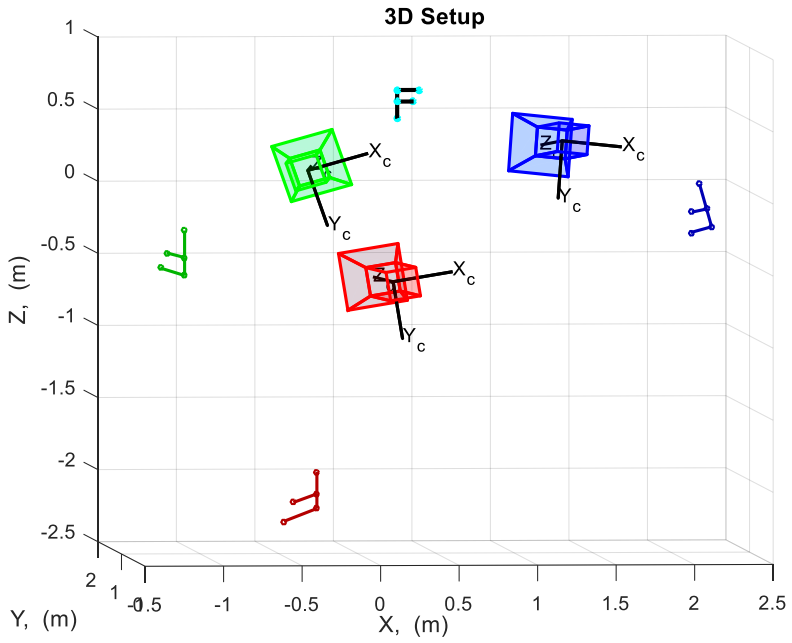


Figura 35 – Riproiezione lettera F, caso generale

3.3.5 – Assunzione sottesa allo sviluppo

Dal punto di vista delle proiezioni-riproiezioni, Il problema è stato evidentemente risolto. **Occorre tuttavia evidenziare la grossa assunzione che si è implicitamente effettuata nello sviluppo del presente algoritmo**. Nello specifico, *si è dato per scontato di conoscere la corrispondenza dei punti rilevati da ciascuna camera ausiliaria*. In altre parole, si è implicitamente assunto che le camere siano state dotate a priori di "un cervello" che consentisse di distinguere, per ogni punto rilevato sul piano di C_1, chi esso sia tra i 5 punti rilevati della camera C_2. Si è dunque sottintesa la conoscenza di un'associazione tra i punti rilevati dai due differenti osservatori ausiliari, cosa non di poco conto. A tal proposito, il *Capitolo 4* verrà dedicato alla discussione del metodo per l'adempimento di tale "mappaggio".

4 – CORRESPONDENCE PROBLEM

4.1 – Matches in Stereo Images – Epipolar Geometry

I programmi presentati fino ad ora sono propedeutici ai programmi per la ricostruzione dell'immagine, tuttavia (come anticipato nel *Capitolo 3*) danno per scontato la conoscenza dei legami tra i punti dei piani immagini delle camere ausiliarie. In altre parole, occorre ora considerare che, non essendo le camere ausiliarie dotate di intelligenza, dato un punto sul piano immagine C_1, non si sa a priori dove esso si trovi nel piano immagine C_2. Tale problema è noto in letteratura con il nome di "*Correspondence problem*". Concettualmente, con riferimento alla *Figura 36*, occorre porsi la seguente domanda:
Come si può, nota l'immagine di C_1, a capire che il punto P_1' corrisponde al punto P_1'' e non P_2''? Se non si tengono in considerazione ulteriori caratteristiche del punto come ad esempio il colore, evidentemente sussistono delle ambiguità.

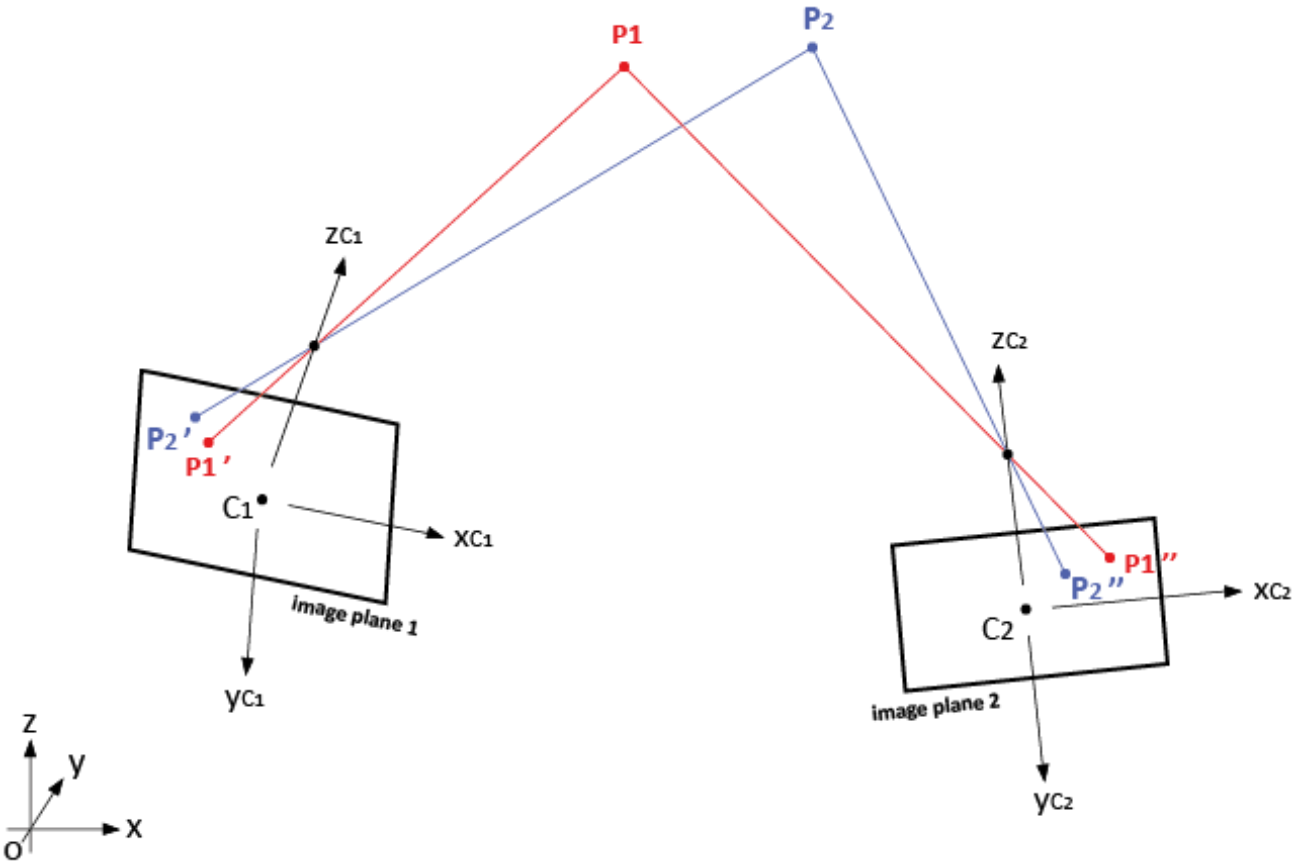


Figura 36 – Correspondence problem: introduzione

Al contrario, in *Figura 37* è illustrato un caso in cui "non vi sono ambiguità": per scovare dove vive P_1 a partire dalla conoscenza delle sue proiezioni su C_1 e C_2, occorre semplicemente cercare sull'immagine di C_2 l'unico pixel "acceso".

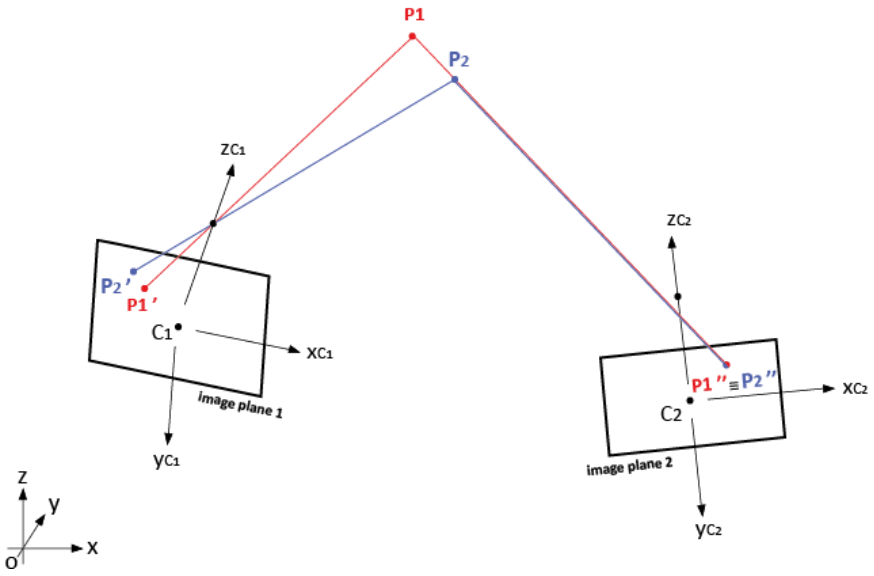


Figura 37 – Correspondence problem: esempio di caso particolare

Si inizia perciò ad intuire che, in un caso più generico di immagini caratterizzate da più pixel accesi e dotati di differente gradazione di colore, vi sia la necessità di un algoritmo capace di individuare le corrispondenze vagliando tali proprietà. É possibile pensare al problema schematizzandolo secondo l'immagine di *Figura 38*:

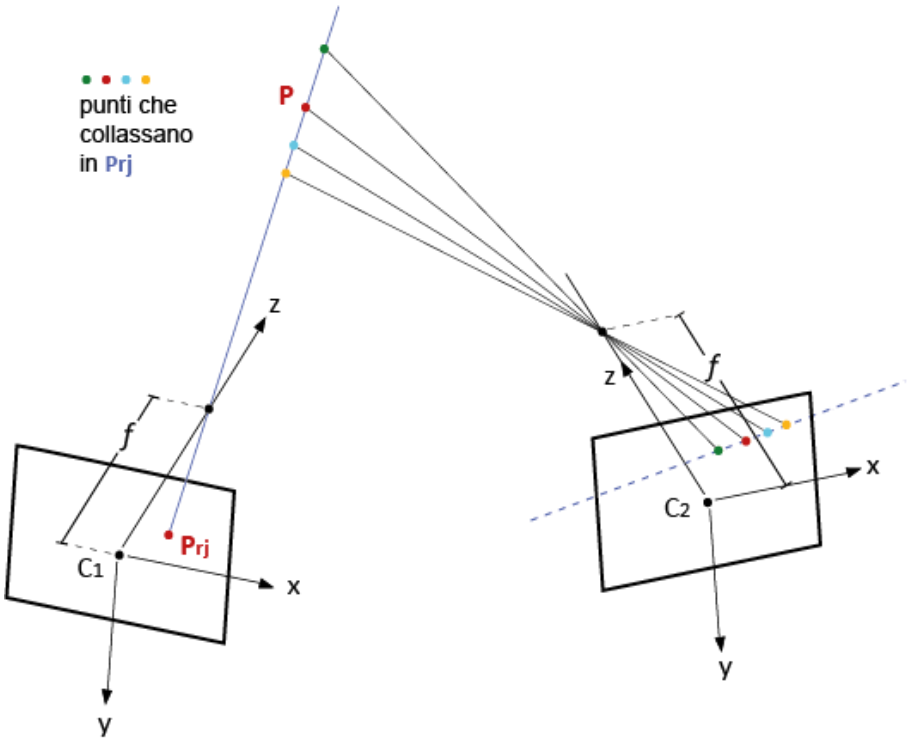


Figura 38 – Correspondence problem: esempio di associazione mediante colori

É di particolare importanza il fatto che ciascun punto della camera C_1 sia geometricamente vincolato ad appartenere ad una data retta del piano C_2: la retta *epipolare*. Tale vincolo, è chiamato *vincolo epipolare* e consente di limitare fortemente il range di recerca delle possibili corrispondenze. Più specificatamente, il problema del "*Matching*" dei punti, che sarebbe a priori un problema di ricerca di corrispondenze da $N \rightarrow N^2$ degenera così ad un caso monodimensionale: $N \rightarrow N$. Il computer che deve eseguire la ricerca ne beneficia in termini di potenza di calcolo richiesta, di contro, occorre evidentemente conoscere preventivamente la posizione relativa delle due fotocamere ausiliarie C_1 e C_2. Dal punto di vista della letteratura, si passa da un problema chiamato *Optical Flow* ad un problema chiamato *Epipolar Geometry*.

4.2 – Disparity Alghorithm e ottenimento del 3D

Se le immagini acquisite dalle camere ausiliarie appartenessero ad uno stesso piano, il problema della ricerca delle corrispondenze e della successiva determinazione dei punti tridimensionali sarebbe matematicamente più conciso del caso generale decritto nel *Capitolo 3*. Per questa ragione, il classico algoritmo di ricerca delle corrispondenze assume che le immagini di input siano complanari, come illustrato in *Figura 39*:

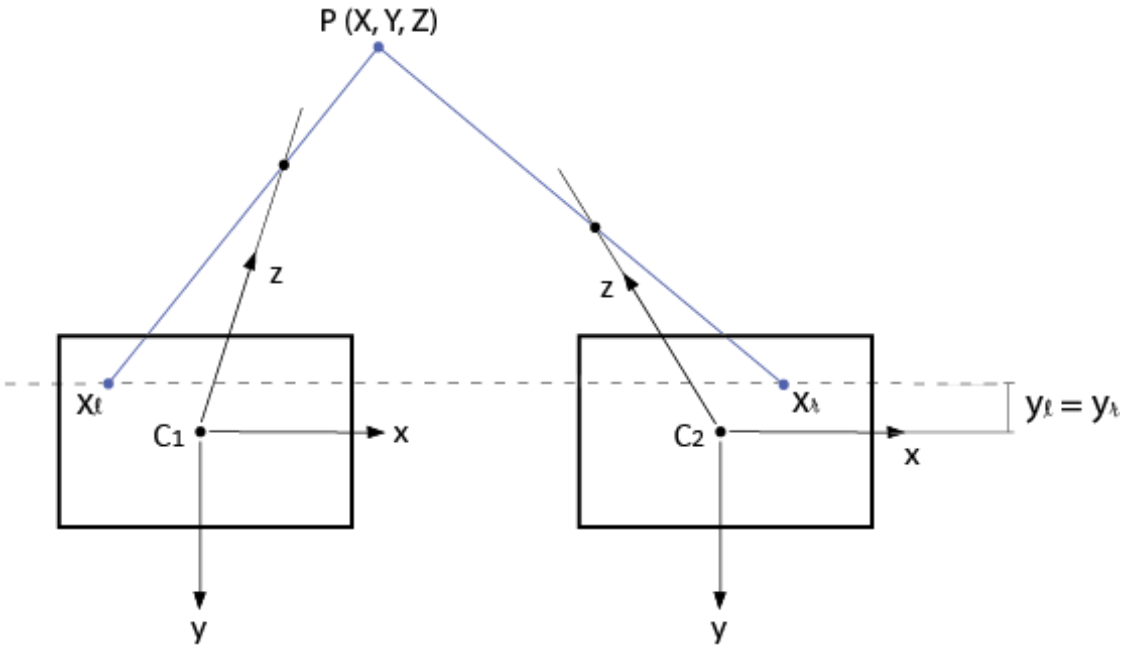


Figura 39 – Schema per la ricerca delle disparità

In questo modo, la ricerca delle corrispondenze si riduce ad una semplice "scansione" secondo linee orizzontali. Sotto questo scenario, la "distanza" (in termini di pixels) tra il punto della camera C_1 considerato ed il corrispettivo nella camera C_2, prende il nome di *disparità*. Nota la disparità tra i due punti, è evidentemente possibile ancora una volta ricostruire la posizione del punto "genitore" P.

Da un punto di vista più concreto, anche volendo fissare le camere C_1 e C_2 secondo lo schema di *Figura 39*, gli inevitabili errori costruttivi o di montaggio fanno sì che i relativi assi ottici non siano perfettamente mutuamente paralleli. Per questa ragione, per disporre di immagini realmente complanari occorrerà manipolare le immagini fisicamente catturate dai due dispositivi in questione. Tale operazione prende il nome di *Rettifica*, e da luogo alle cosidette "*Rectified stereo pair images*". Queste ultime, come anticipato, saranno l'input funzionale dell'algoritmo di ricerca delle disparità.

Dopo questo antefatto, ci si accinge alla descrizione dell'algoritmo di disparità vero e proprio.

Una delle idee più immediate per determinare la corrispondenza tra i punti dell'immagine di sinistra ed i punti dell'immagine di destra, è quella di considerare, per ogni punto dell'immagine di sinistra, un'intorno adeguatamente ampio e confrontare questa "finestra d'osservazione" con tutte le finestre dei punti della camera di destra che giacciono sulla linea epipolare. La finestra di destra che "assomiglia maggiormente" a quella di sinistra, è associata al punto corrispondente ricercato.

Si assume quindi una "funzione di diversità", che associa a ciasun punto dell'immagine di sinistra una n-upla di valori, con n numero di pixels in orizzontale. Questi valori sono indice della diversità tra le varie coppie di finestre d'osservazione, e l'obiettivo è cercare il minimo di questa funzione. Tale minimo cadrà in corrispondenza del punto della camera di destra che sia il "meno diverso" tra le varie coppie analizzate. Esso è rappresentativo della corrispondenza che si sta cercando. Il tutto è quindi naturalmente ricondotto ad un problema matematico, in particolare ad un problema di confronto numerico tra le varie finestre, dette "*windows*".
Si assuma che l'immagine catturata dalla camera di sinistra ("*left*") e da quella di destra ("*right*") abbiano la medesima risoluzione. In altre parole, siano le matrici associate a ciascuna immagine, della stessa dimensione. Si assuma inoltre che l'immagine sia stata preventivamente rettificata e trasformata in scala di grigi. Allora, la formula matematica che si sta assumendo è del tipo:

$$SAD_i(j) = \sum_{k=1}^{nP_W} \left(x_{l,k}(i) - x_{r,k}(j) \right)^2$$

A secondo membro la sommatoria è estesa ad un solo indice (k) per lo scorrimento sia lungo le righe che lungo le colonne della finestra. Ciò è lecito in virtù dell'assunzione di ordinamento lessicografico dei vari punti componenti la finestra d'osservazione stessa.

Gli altri elementi sono:

- i, numero del pixel considerato nell'immagine di sinistra. $i = 1,2, \dots, nColu$; $nColu =$ numero di colonne della matrice associata all'immagine
- j, numero del pixel considerato nell'immagine di destra. $j = 1,2, \dots, nColu$
- nP_w, numero di pixels di ciascuna finestra (*window*) d'osservazione
- $x_{l,k}(i)$, valore numerico del pixel k-esimo nell'immagine di sinistra
- $x_{r,k}(j)$, valore numerico del pixel k-esimo nell'immagine di destra, associato alla j-esima finestra d'osservazione

Si riporta in *Figura 40* un esempio grafico particolarmente agevole per l'applicazione del suddetto algoritmo, rinominato tra gli script Matlab allegati "*DisparityAlghorithm*".
Siano date le due immagini della figura successiva, scattate rispettivamente dalla camera di sinistra, C_1, e da quella di destra, C_2:

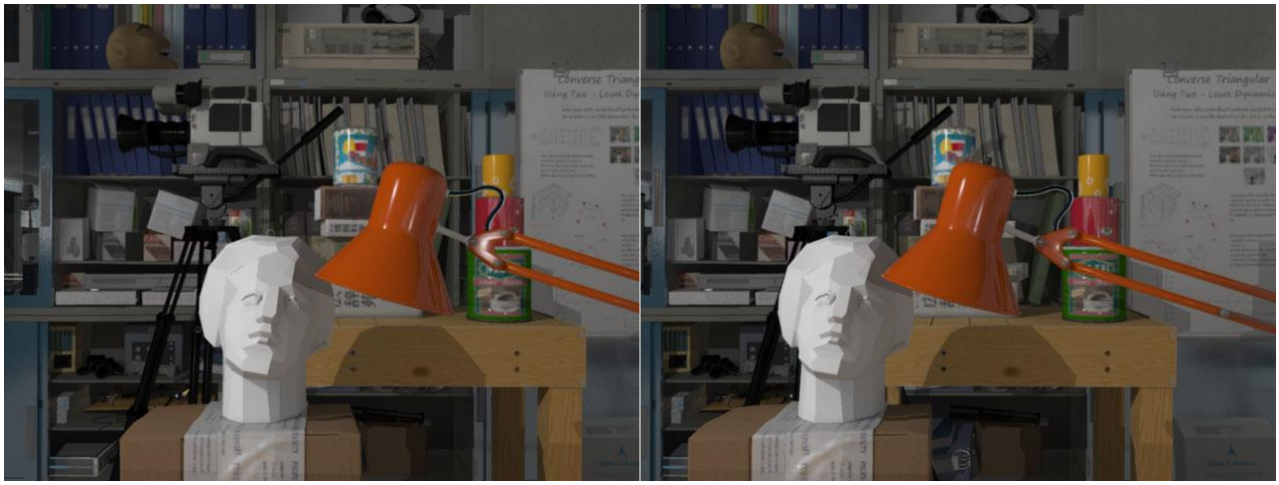


Figura 40 – Coppia di immagini stereo rettificate (Immagine dell'università Tsukuba)

L'immagine di *Figura 41* riporta un'esempio applicativo grafico dell'algoritmo appena descritto. Si noti che la distanza tra le varie finestre d'osservazione è stata volutamente maggiorata, per maggior chiarezza espositiva.

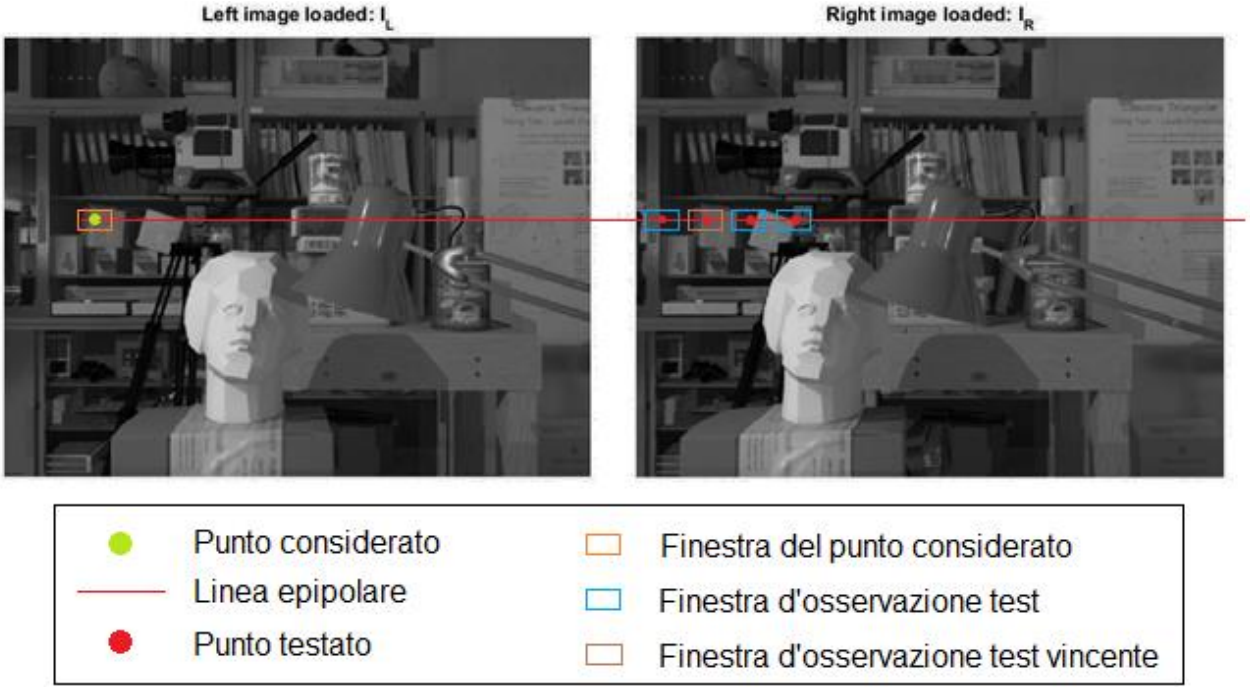


Figura 41 – Esempio di ricerca della disparità, algoritmo SAD

Per il punto dell'immagine di sinistra considerato, si osserva (*Figura 42*) che il minimo della funzione *SAD* giace al valore 6. Questo implica che, di tutte le finestre scandagliate nell'immagine di destra, la sesta è quella dotata di valori di gradazione di colore più simili alla finestra di sinistra.

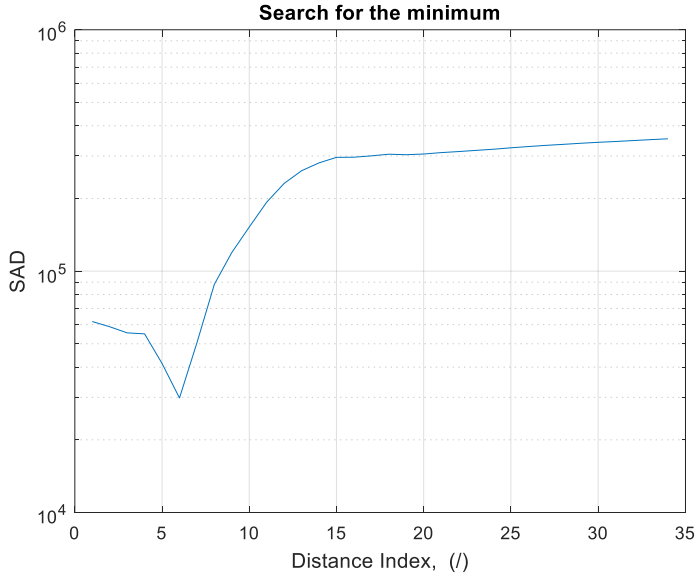


Figura 42 – SAD del punto considerato

Si conclude così la ricerca della corrispondenza per detto punto dell'immagine di sinistra
Si noti che non si sono scandagliate tutte le possibili finestre d'osservazione lungo l'epipolare: oltre un certo range di pixels (in questo caso si è assunto il valore 35) è inutile andare a cercare poiché non ci si aspetta che la disparità assuma valori eccessivamente grandi.

Facendo questa operazione per ogni punto, si viene a definire la cosidetta "*Mappa delle disparità*", ovvero una matrice in cui la posizione dell'elemento è associata alla riga e colonna del pixel in questione, ed il valore ne rappresenta la distanza (in pixel) tra i due corrispondenti punti delle due immagini. Associando valori sempre più chiari al crescere della disparità, è possibile generare un'immagine del tipo quella illustrata in *Figura 43*:

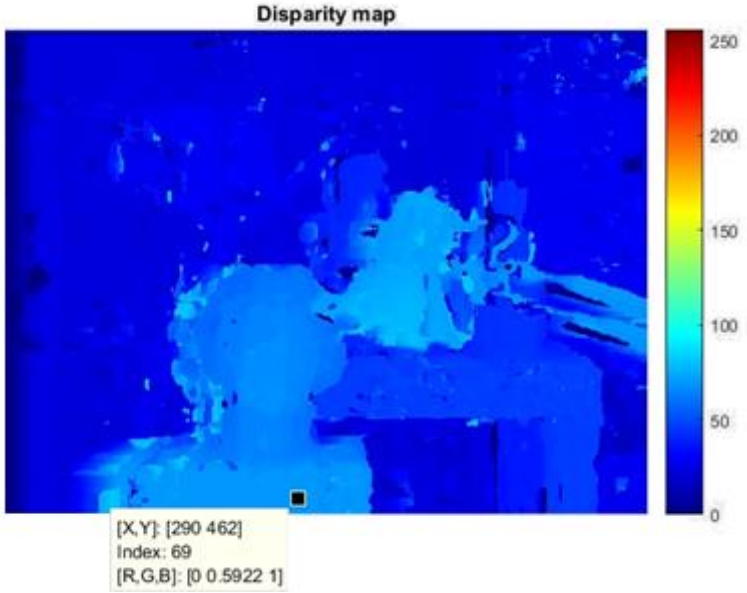


Figura 43 – Mappa delle disparità previa semplice applicazione dell'algoritmo SAD

Essa altro non è che una prima rappresentazione della profondità dell'immagine: nota la disparità è infatti possibile risalire alla profondità mediante semplici relazioni di natura geometrica.

Si opta nel seguito per utilizzare l'algoritmo sotteso all'interno della funzione Matlab "*disparity*" sviluppato in collaborazione con *IBM*, in quanto matematicamente più avanzato e veloce. Non essendo l'obiettivo della presente trattazione di esporre i vari metodi per la stima della disparità, non verranno spese ulteriori parole a riguardo: ci si limita solo ad effettuare un confronto visivo tra i risultati proposti dai due algoritmi e a descrivere macroscopicamente il metodo utilizzato da Matlab.

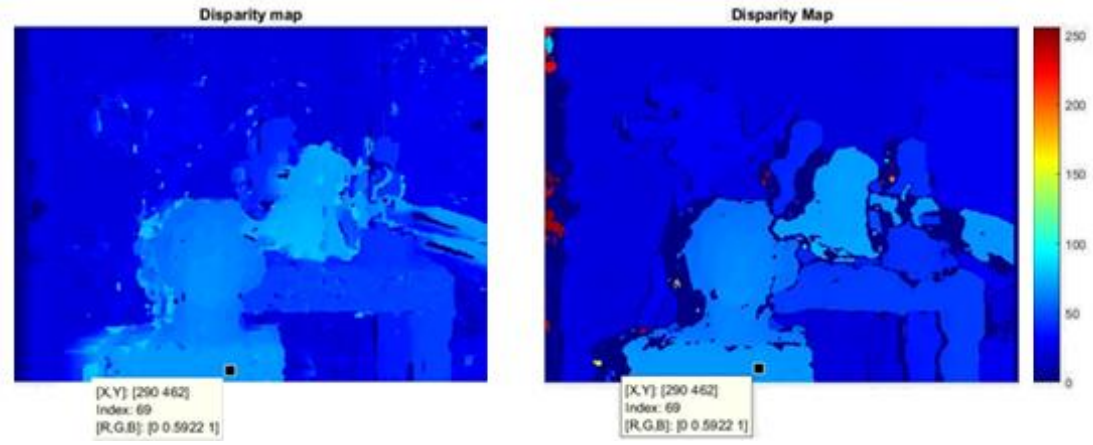


Figura 44 – Mappa delle disparità. Sinistra: SAD. Destra: algoritmo by Matlab & IBM

Sulla sinistra, il semplice algoritmo SAD. Sulla destra, i risultati proposti da Matlab mediante l'utilizzo della funzione *disparity*. Si noti che in quest'ultimo caso, laddove i valori numerici trovati sono risultati anomali, vengono scartati ponendoli pari "ad un meno infinito" (numerico).

La funzione Matlab *disparity* implementa due differenti algoritmi: Confronto tra blocchi e Confronto tra blocchi semi-globale. Questi algoritmi consistono nell'applicazione in sequenza dei successivi tre steps:

- Calcolo di una misura dei contrasti dell'immagine, utilizzando un Sobel filter
- Calcolo delle disparità per ogni pixel dell'immagine di sinistra
- Assegnamento di un dato valore agli elementi per i quali il calcolo della disparità non è andato a buon fine. Tale valore è imposto pari a "-REALMAX('single')"

La mappa delle disparità costituisce una vera e propria anteprima della scena tridimensionale: disparità e profondità (z) sono indissolubilmente legate, in particolare l'una è inversamente proporzionale all'altra. La *Figura 45* mostra la mappa di disparità in cui, per chiarezza espositiva, ad ogni colore viene attribuita una certa altezza:

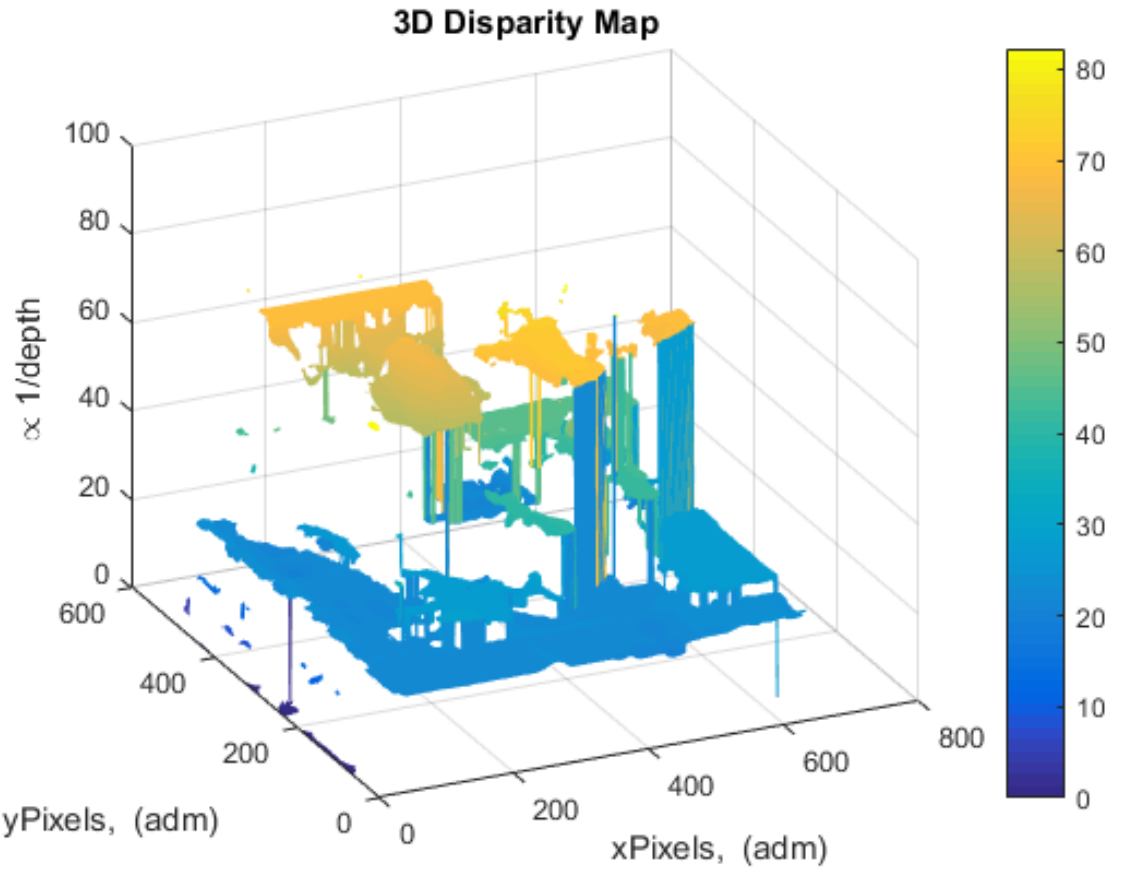


Figura 45 – Mappa delle disparità tridimensionale

Ad esso corrisponde la scena tridimensionale illustrata in *Figura 46*:

Figura 46 – Estrazione scena tridimensionale, caso input immagine virtuale

Il principale problema che si evidenzia in questa sede di progetto è la presenza delle occlusioni. Un'occlusione si manifesta quando, in virtù della particolare disposizione del mondo tridimensionale e delle camere, certi oggetti vengono visti e catturati da una camera ma non dall'altra. In questo caso, gli algoritmi di confronto del tipo SAD, non possono effettuare delle valutazioni coerenti in quanto cercherebbero un qualcosa che non c'è nei dati (questo spiega le consistenti "sfumature" nell'intorno della statua di *Figura 43*).
E' possibile osservare che ciò che di fatto esegue la funzione Matlab *disparity* è una rimozione di questi punti particolarmente ambigui, considerandoli come "*outlayers*". Ciò comporterà però il successivo limite di non riuscire a ricostruire correttamente tutta la scena. E' tuttavia possibile pensare di migliorare la situazione utilizzando un sistema di visione a 3 o più camere, sebbene computazionalmente ed economicamente parlando sia più dispendioso.

5 – IMAGE SINTHESYS, NO REAL TIME

5.1 – Logica di programmazione

Note le trasformazioni omogenee tra le due fotocamere ausiliarie, è possibile ottenere in modo immediato l'allineamento di qualsiasi coppia di immagini da esse acquisite. Ciò è valido per qualsiasi orientamento relativo tra le camere. Si tenga presente che è possibile stimare l'allineamento delle due immagini anche in assenza dei parametri appena esposti, ma ciò è temporalmente più dispendioso e probabilmente meno preciso. Per questa ragione, si opta per eseguire una calibrazione delle camere (una volta sola) a monte.

Allineate le immagini acquisite dalle fotocamere mediante il processo di rettifica, è possibile eseguire la stima della mappa delle disparità, e ottenere conseguentemente la scena 3D.

Nota la scena 3D, è possibile riproiettare i punti del mondo sul piano immagine dell'osservatore C_0, realizzando così l'immagine sintetizzata.

Lo schema logico adottato nel seguito è riassunto in *Figura 47*:

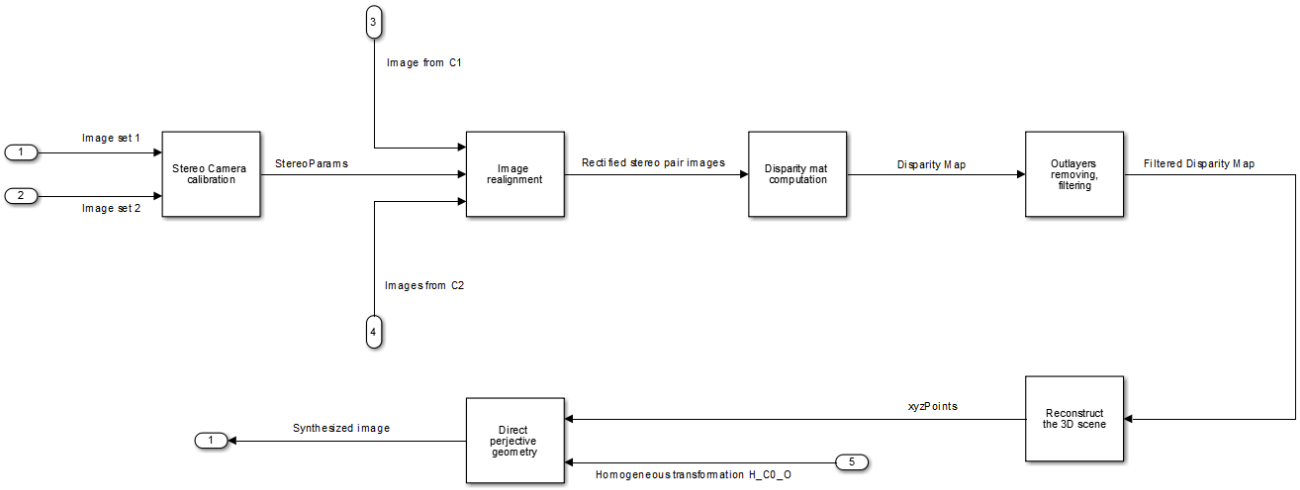


Figura 47 – Schema a blocchi per la sintesi dell'immagine, No Real-time

5.2 – Test low cost A, fotocamere qualsiasi

Per la calibrazione preventiva delle fotocamere, si vuole realizzare una *Checkerboard* con un budget a disposizione di 10 €. Si sceglie di attribuire come taglia di ciascun quadrato le dimensioni di 100x100 mm^2. La *Checkerboard* così concepita è mostrata in *Figura 48*:

Figura 48 – Realizzazione prima CheckerBoard

Mediante materiale di recupero, si sceglie di costruire una piattaforma per il sostegno delle tre fotocamere. Non sapendo a priori quale fosse la configurazione più intelligente per il fissag gio

delle fotocamere, si è realizzata una base che consentisse diversi settaggi. Tale struttura è illustrata in *Figura 49*:

Figura 49 – Primo setup sperimentale: base di sostegno per le fotocamere

Il processo di calibrazione automatico fornisce il seguente *Warning*, mettendo in luce il fatto che sarebbe stato meglio aggiungere/rimuovere una riga o colonna di quadrati alla scacchiera della CheckerBoard:

> Warning: The checkerboard must be asymmetric.
> One side should be even, and the other should be odd.
> Otherwise, the orientation of the board may be detected incorrectly.

Nonostante questo messaggio di aviso, l'algoritmo funziona correttamente e pertanto si opta per ignorare momentaneamente tale suggerimento.

In questo caso, è stato utilizzato un *Nokia Lumia 1020* come camera di sinitra, rinominato C_0, ed il *Samsung Galaxy S6 Edge* a destra, rinominata C_1.
Il set di immagini acquisite è illustrato in *Figura 50*:

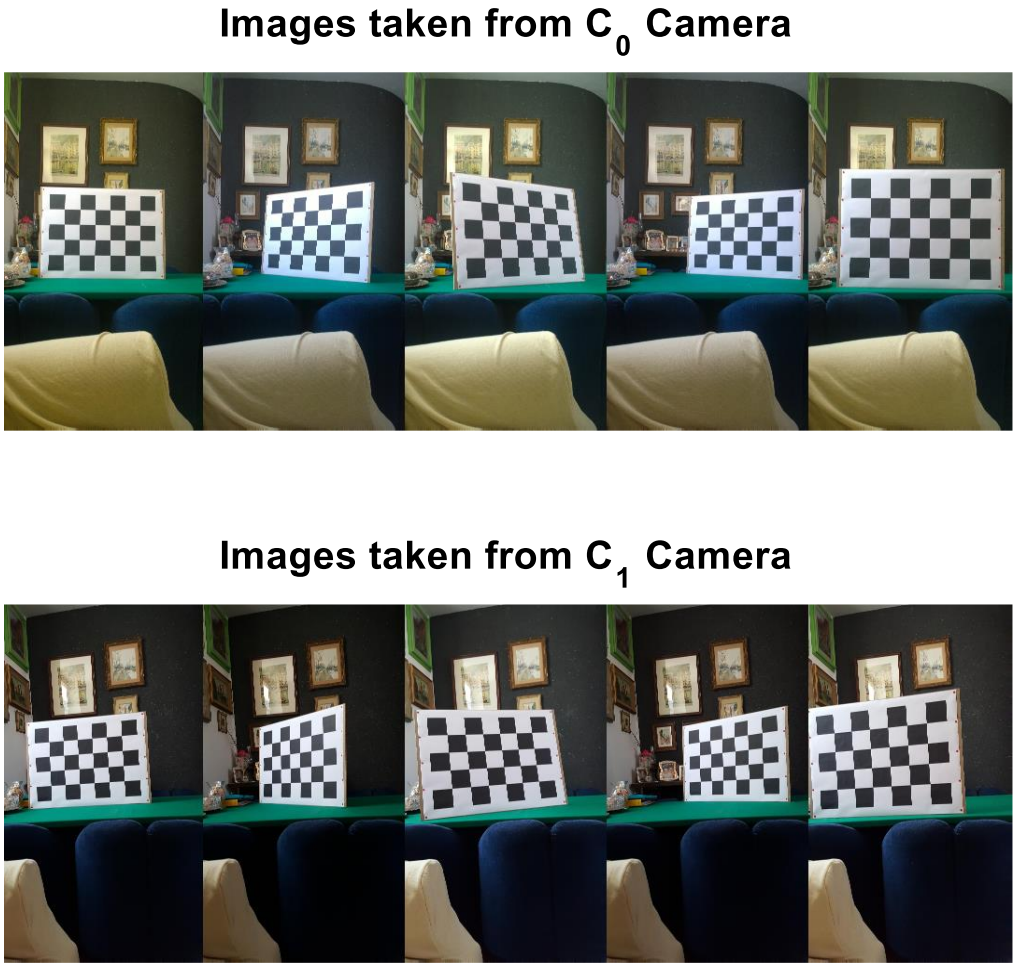


Figura 50 – Primo test: immagini di calibrazione con Nokia e Samsung

Mediante l'algoritmo di ricerca automatica dei vertici della Checkerboard, si ritrovano i risultati di *Figura 51*:

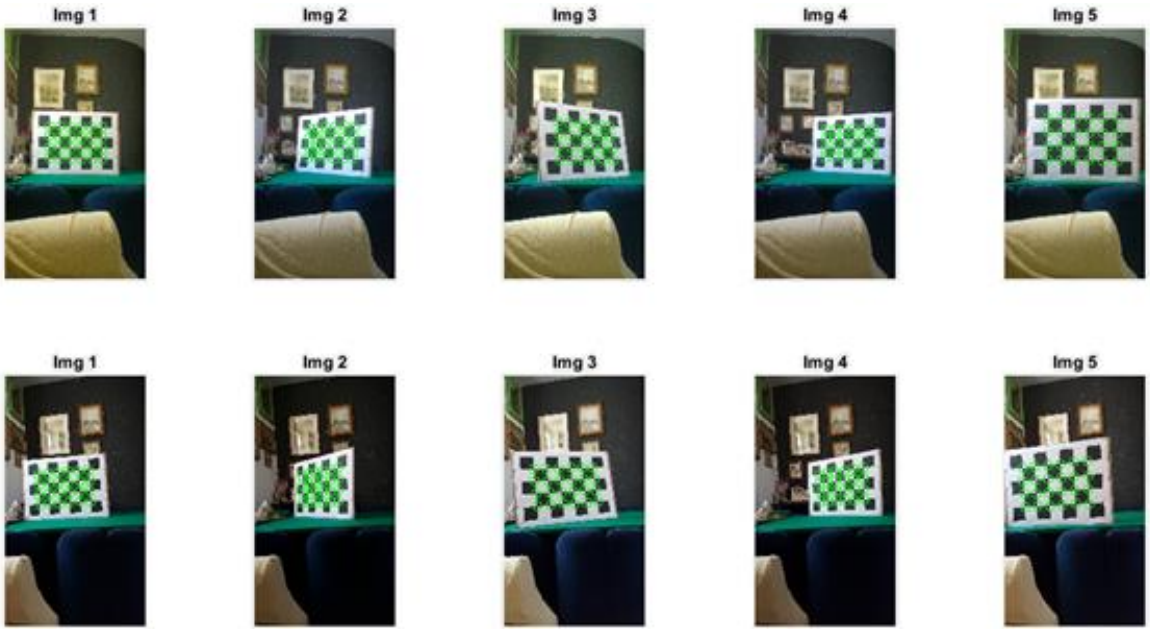


Figura 51 – Primo test: immagini di calibrazione con Nokia e Samsung, rilevamento punti

In *Figura 52*, l'errore medio sulla riproiezione dei punti. Ciascuna colonna è rappresentativa della diversità tra i punti rilevati sulla Checkerboard dell'immagine in questione ed i punti riproiettati su di essa sulla base dei parametri della camera ottenuti calibrando.

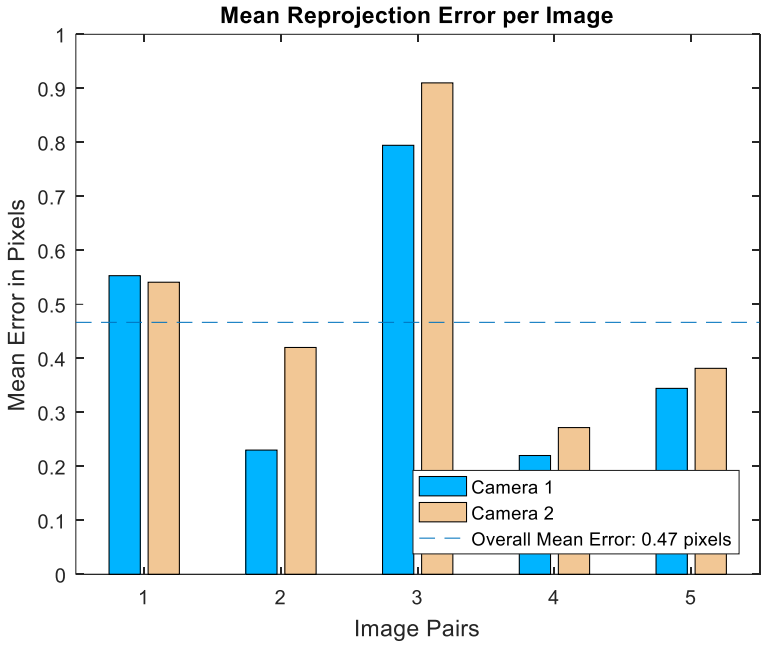


Figura 52 – Primo test: errore di riproiezione sulle immagini di calibrazione, Nokia e Samsung

In *Figura 53*, l'interpretazione visiva della geometria del setup.
Si presti attenzione a non confondere nella seguente immagine (e nelle successive analoghe) il numero 1 e 2 attribuiti alle camere dalla grafica di Matlab con i nomi di C_0, C_1 o C_2 attribuiti nel corso della programmazione successiva. Nel resto della trattazione si darà per scontato che il lettore sappia orientarsi in base al contesto ed attribuire il giusto nome alle camere ausiliarie ed all'osservatore nascosta.

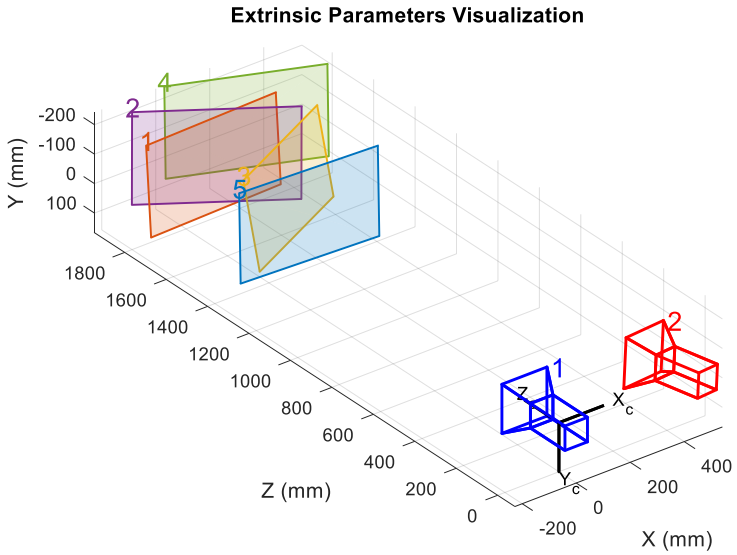


Figura 53 – Primo test: visualizzazione settaggio delle camere, Nokia e Samsung

In *Figura 54*, un confronto visivo dei punti rilevati e quelli riproiettati. Questi ultimi, sono figli del contributo di dati ottenuti dalla valutazione *dell'intero set di immagini* nel suo complesso, conseguentemente non potranno mai concidere perfettamente con quelli rilevati nella singola immagine.

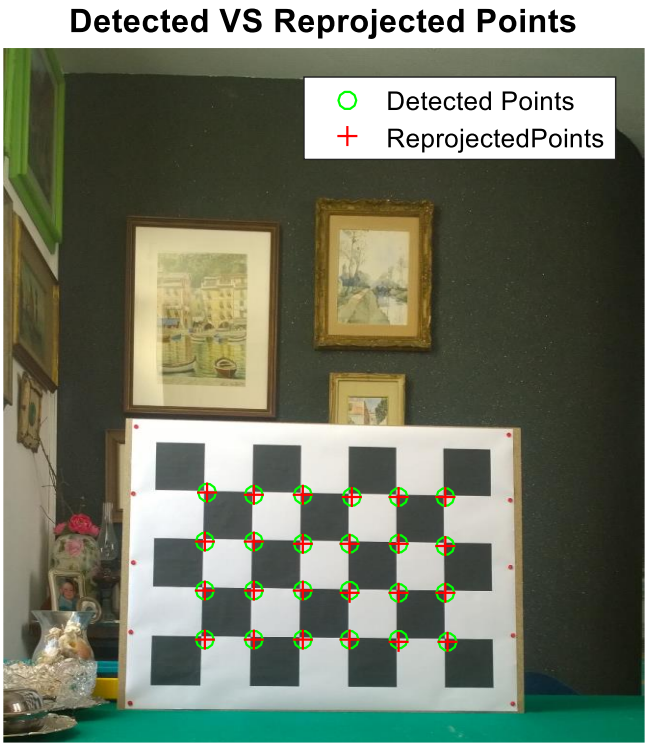


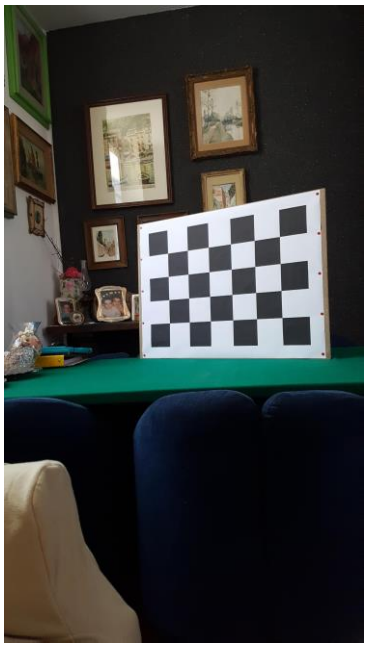
Figura 54 – Primo test: interpretazione visiva dei punti rilevati VS riproiettati, Nokia e Samsung

Si farà ora riferimento all'algoritmo descritto in precedenza per la sintesi dell'immagine.

In *Figura 55* sono illustrate le immagini catturate dalle due fotocamere, a partire dalle quali si tenterà di ricostruire la scena tridimensionale.

É possibile osservare anzitutto che, avendo collocato le fotocamere ausiliarie al 3D molto distanti e sfasate reciprocamente, vi è una consistente diversità tra gli oggetti inquadrati da esse. É dunque possibile che l'algoritmo della realizzazione della mappa delle disparità non funzioni correttamente o comunque abbia dei vuoti consistenti. Questo in virtù del fatto che andrà a cercare, durante l'algoritmo di corrispondenza, oggetti che fisicamente non ci sono.

Captured scene by C_1



Captured scene by C_2

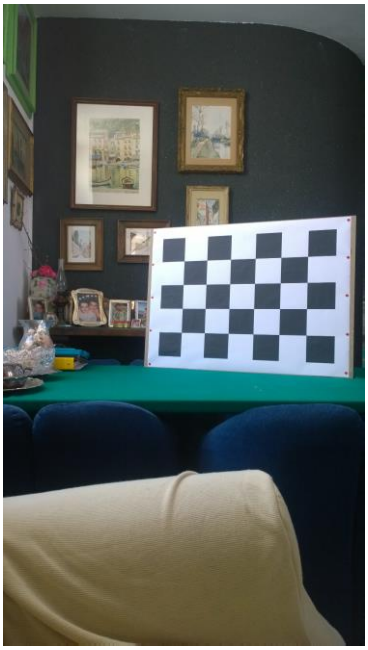
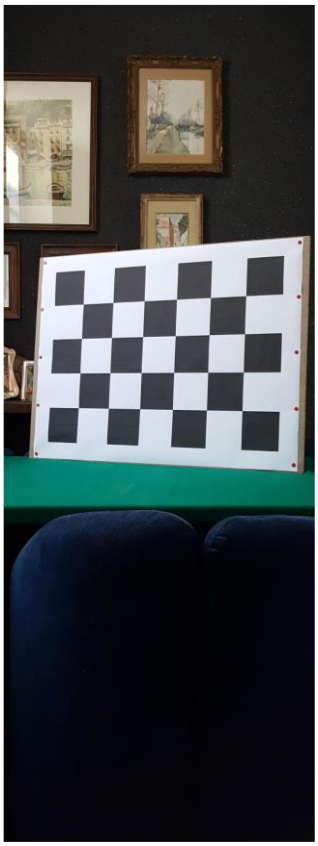


Figura 55 – Primo test: immagini input per generazione 3D, Nokia e Samsung

Di seguito l'output delle immagini dopo il riallineamento:

Realigned image from C_1



Realigned image from C_2

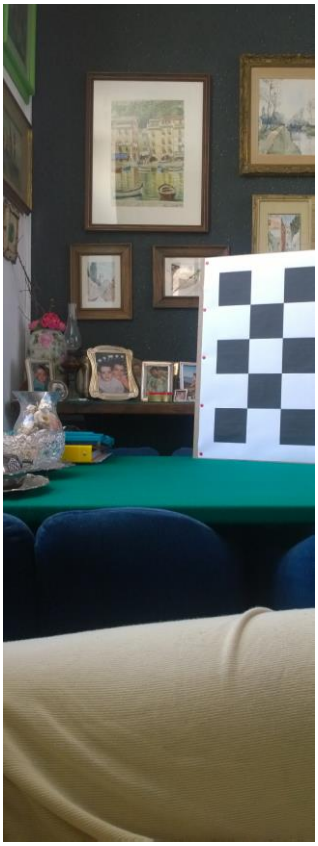


Figura 56 – Primo test: immagini rettificate per generazione 3D, Nokia e Samsung

Come anticipato, la diversità dei due punti d'osservazione ha creato problemi non indifferenti anche durante la sola fase di reallineamento dell'immagine: tantissimi oggetti presenti in un'immagine riallineata non ci sono all'interno dell'altra immagine riallineata.
Inoltre, la notevole diversità tra i due punti d'osservazione impone che i riflessi di luce siano totalmente spostati tra un'immagine e l'altra. Conseguentemente, l'algoritmo di disparità che prenderà in pasto queste immagini, non potrà che lavorare male.

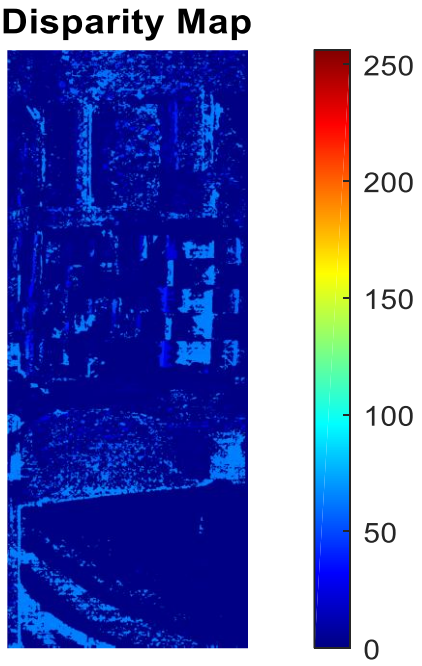


Figura 57 – Primo test: mappa delle disparità, caso reale, Samsung e Nokia (Esito fallimentare)

La mappa delle disparità è totalmente inopportuna, pertanto non si opta per non ricostruire la scena tridimensionale nonchè la successiva immagine sintetizzata.
Ci si limiterà in questa sede a trovare qualche altra spiegazione sulle possibili cause di malfunzionamento.
Anzitutto, al di la dell'input che già di per se è inadeguato, occorre considerare che i parametri di input dell'algoritmo di disparità sono stati settati al valore di default.
Tra tutti questi parametri, uno dei più importanti (in questo specifico caso) è il disparityRange, ovvero il range di pixels in cui cercare la corrispondenza tra un'immagine e l'altra. Il valore di default scelto da *Mathworks* è [0 255], che significa che, preso un punto dell'immagine di sinistra, la corrispondenza nell'immagine di destra andrà ricercata a partire dalla coordinata di quel pixel, sino a 255 pixels più a destra. Poichè tuttavia le due fotocamere sono state montate in modo che l'asse ottico tenda a convergere, è possibile che la corrispondenza tra i punti dell'immagine di sinistra e quella di destra, vada cercata anche tra i pixels a sinistra del corrisettivo.
Un ulteriore possibile fonte di malfunzionamento è il fatto di avere utilizzato due fotocamere troppo differenti tra esse. Per come funziona l'algoritmo di disparità, esso andrà a cercare corrispondenze che in termini di colore (al di la della diversa locazione delle camere) non possono essere riscontrate, in virtù della diversità fisica dei due sensori. Ciò può essere parzialmente compensato settando ulteriori parametri di input della funzione *disparity*, tuttavia vi è l'evidenza

di un problema ancora più grave. Si tratta del diverso "sgranamento" che hanno le immagini catturate dai due differenti dispositivi, e ciò non può che far fallire la ricerca di molte corrispondenze. Probabilmente tale differenza non è molto percepibile stampata su carta, ma dinnanzi ad un display ce ne si convincerebbe facilmente. Le immagini di *Figura 58* tentano di esprimere questi ultimi concetti prendendo a riferimento dei punti di particolare interesse. Si tenga presente che comunque il tutto avviene a livello di scala di grigi, e non dei tre canali *RGB*.

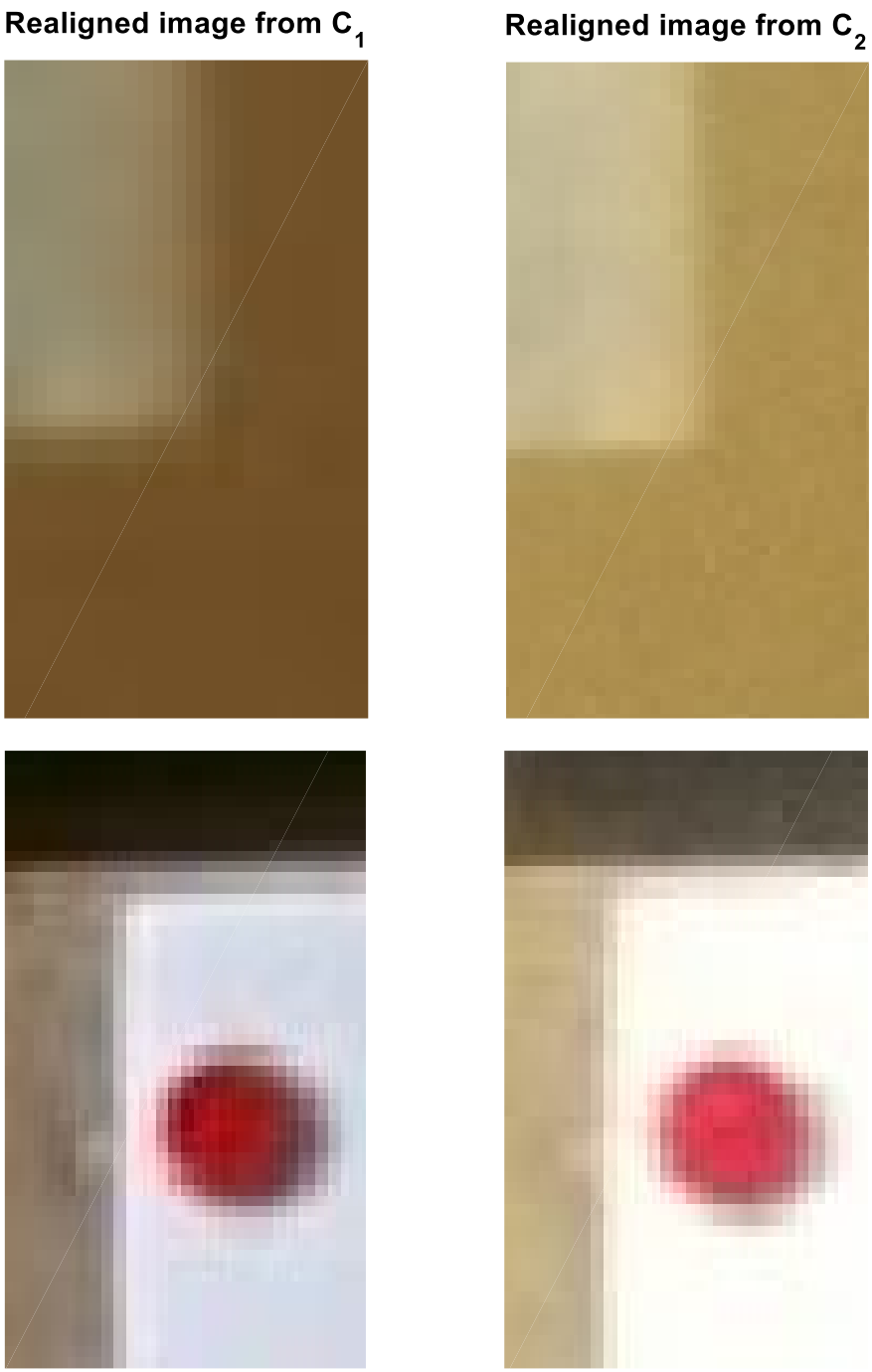


Figura 58 – Effetto nocivo dell'utilizzo di fotocamere molto differenti: esempio

Per quanto detto, il test viene già considerato fallito e pertanto non si tenta di ricostruire il 3D.

5.3 – Test low cost B, fotocamere qualsiasi, cambio posizione

In questa sede, si prova a ridurre il problema delle occlusioni e dei cambi di riflesso avvicinando i due smartphone e rendendoli circa paralleli. La situazione è analoga a quella illustrata in *Figura 59*:

Figura 59 – Secondo setup sperimentale, camere Nokia e Samsung

Il rillievo dei punti sulle fotografie in ingresso è mostrato in *Figura 60*:

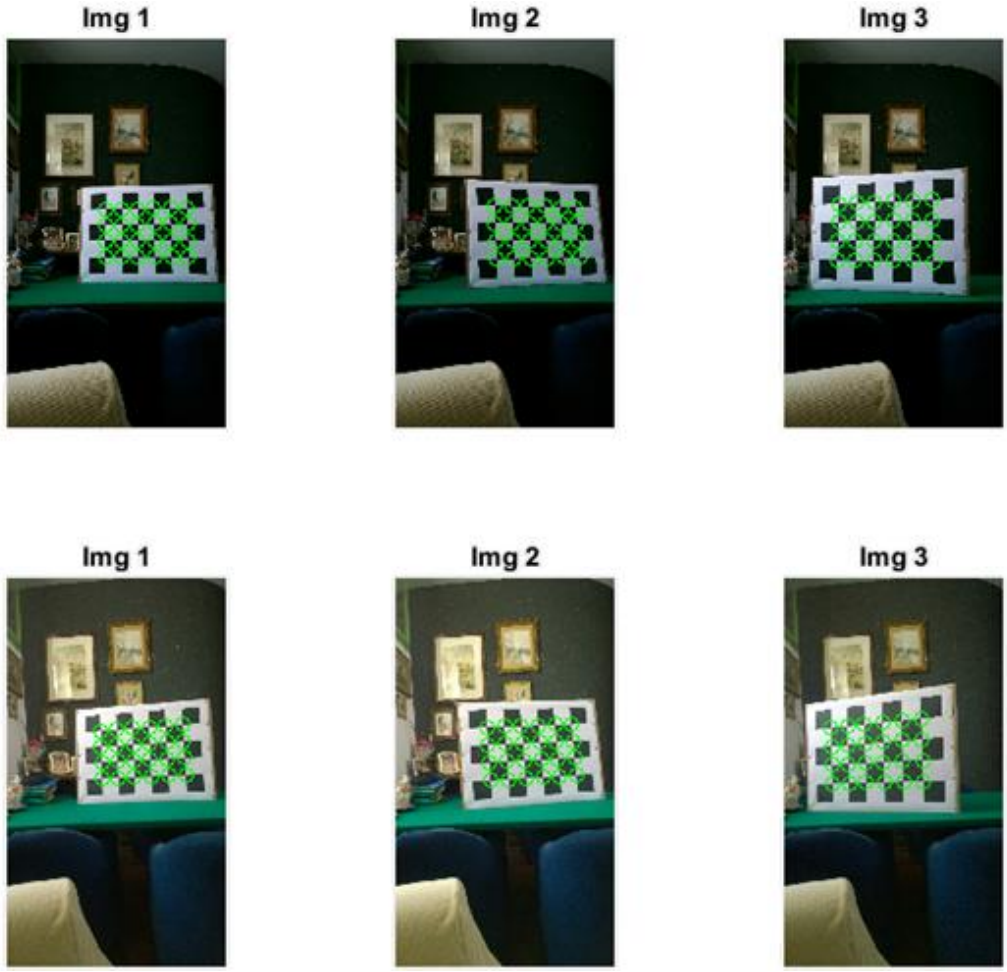


Figura 60 – Secondo test: immagini di calibrazione con Nokia e Samsung, rilevamento punti

In *Figura 61* la visualizzazione dei nuovi parametri estrinseci delle camere:

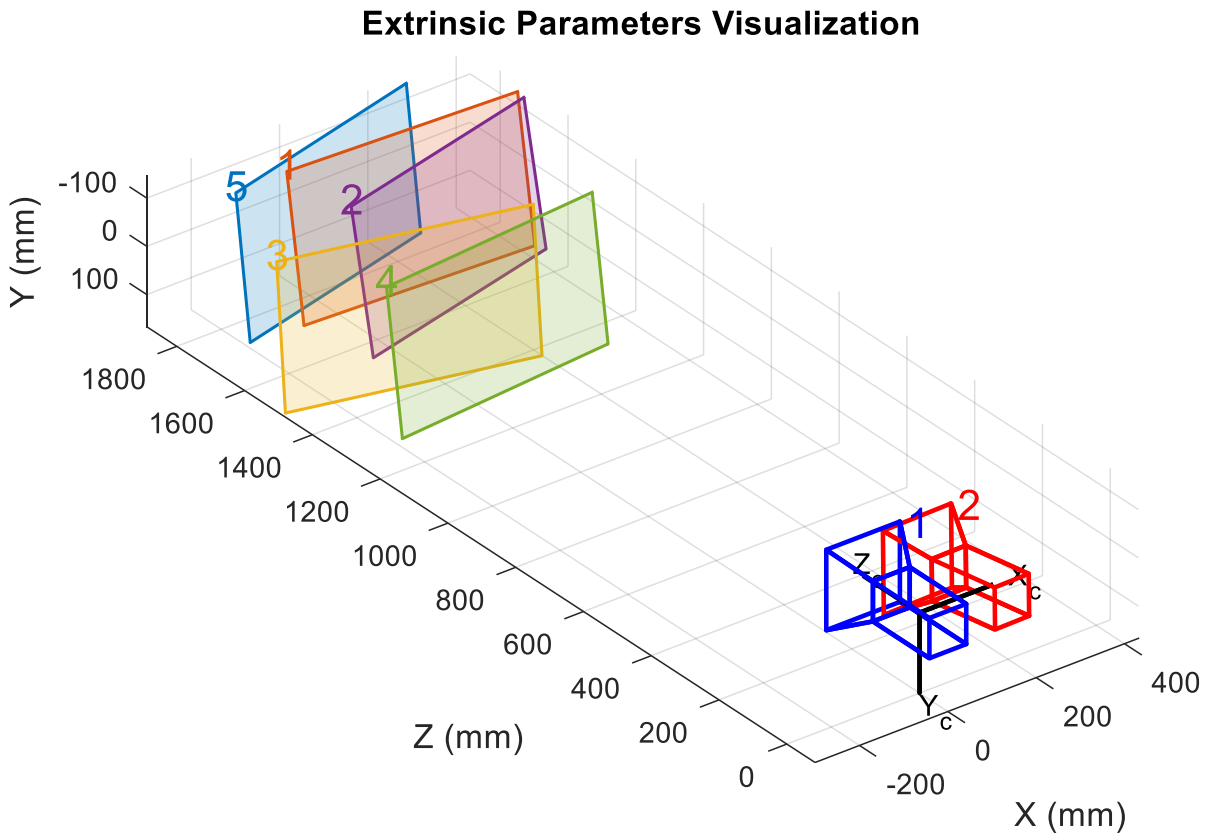


Figura 61 – Secondo test: visualizzazione settaggio delle camere, Nokia e Samsung

In *Figura 62* il dettaglio della posizione ed orientamento relativo delle due camere utilizzate:

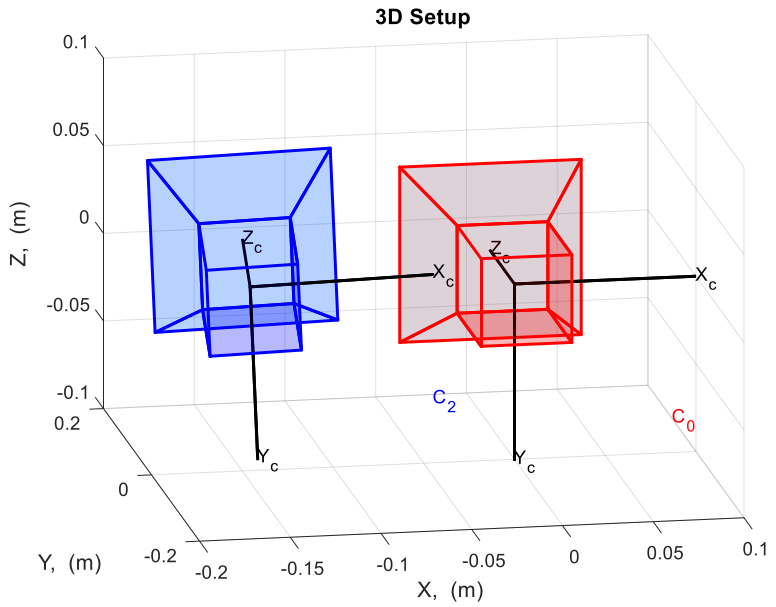
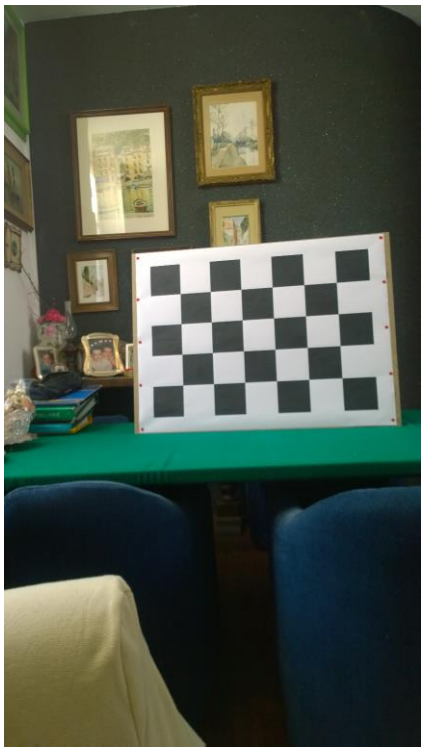


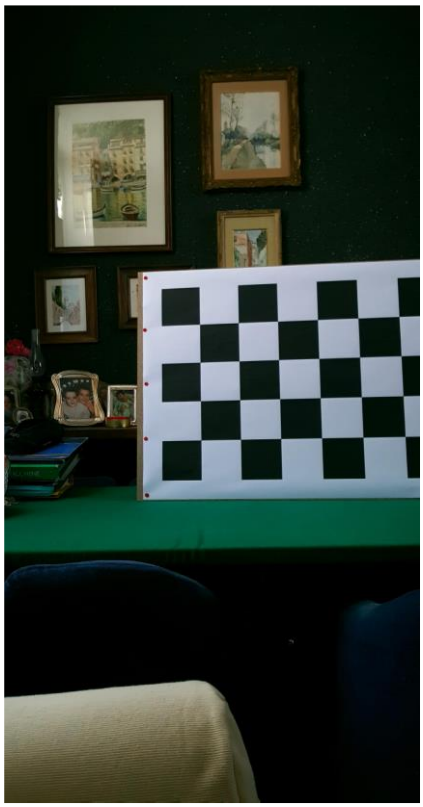
Figura 62 – Secondo test: dettaglio posizione ed orientamento camere, Nokia e Samsung

Captured scene by C_1

Captured scene by C_2



Realigned image from C_1



Realigned image from C_2

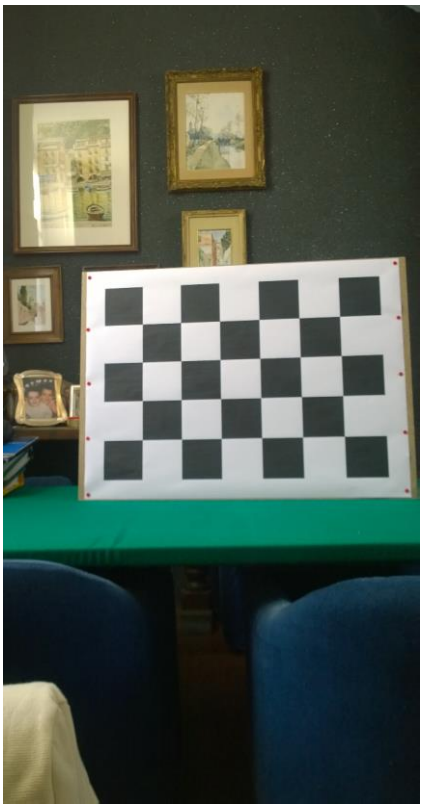


Figura 63 – Secondo test: immagini input e rettificate per generazione 3D, Nokia e Samsung

Si è così ottenuta la mappa delle disparità mostrata in

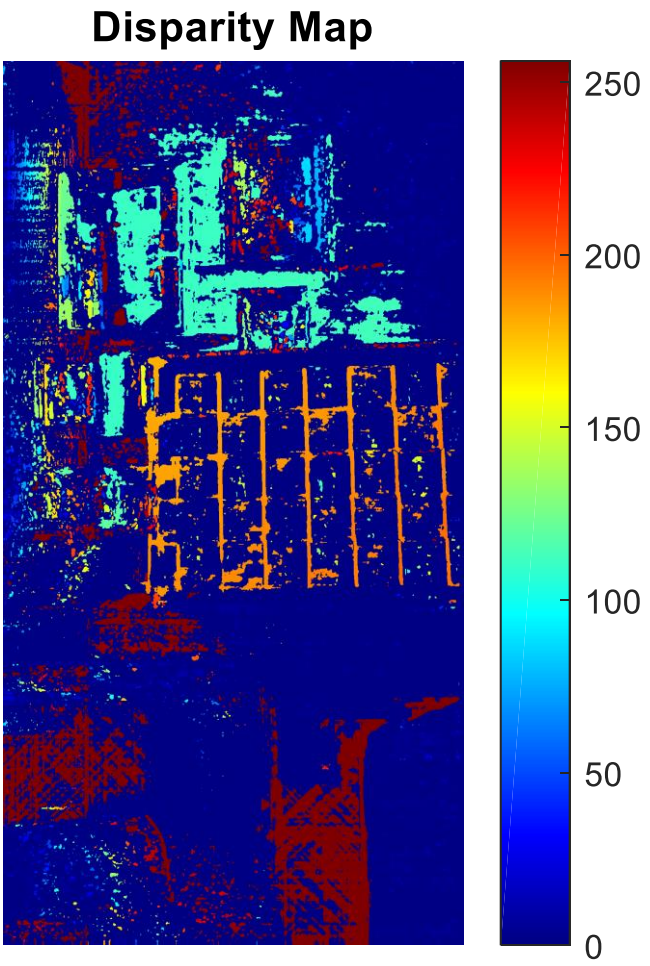


Figura 64 – Secondo test: mappa delle disparità, caso reale, Samsung Nokia (Esito fallimentare)

Anche in questo caso, nonostante gli svariati tentativi di settare differenti valori di parametri ad input della funzione disparity, l'output è di qualità insufficiente.

Ancora una volta, la sintesi dell'immagine è da considerarsi fallita ed il tentativo di ricostruzione del 3D viene abbandonato in favore di una modifica consistente del setup sperimentale.

5.4 – Test low cost C, fotocamere stesso modello, Indoor

L'idea attuale è quella di ridurre tutte le possibili fonti di errori riscontrati nei test precedenti, utilizzando due telefoni dotati della "medesima" fotocamera. Per questa ragione, si è reperito in prestito un altro *Samsung Galaxy S6 Edge*, e lo si è collocato affiancato al primo, come nella maggior parte delle stereo camere nonché come avviene nel sistema visivo umano. Il nuovo setup sperimentale è mostrato in *Figura 65*:

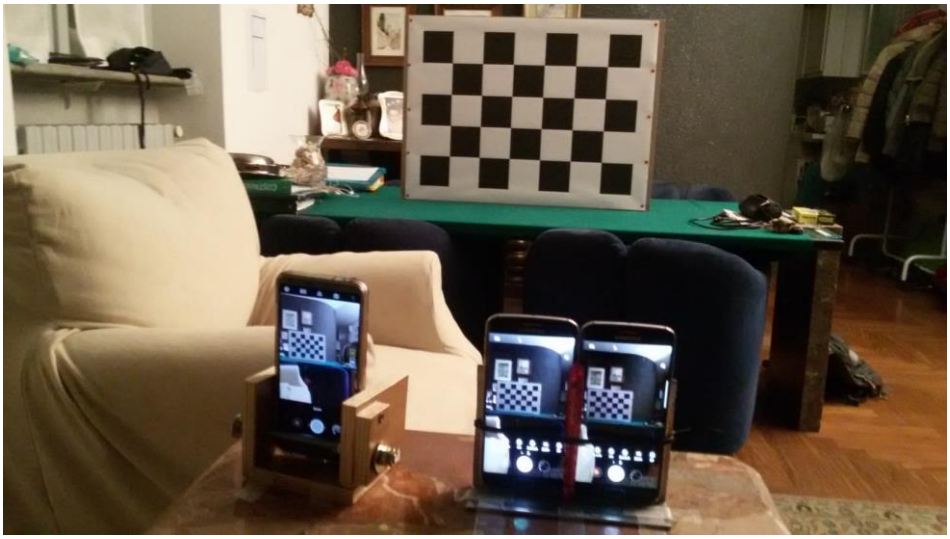


Figura 65 – Terzo setup sperimentale, camere Samsung e Samsung

Come al solito, ci si accinge alla calibrazione delle fotocamere. Si acquisiscono dai due *Samsung* le immagini mostrate in *Figura 66*:

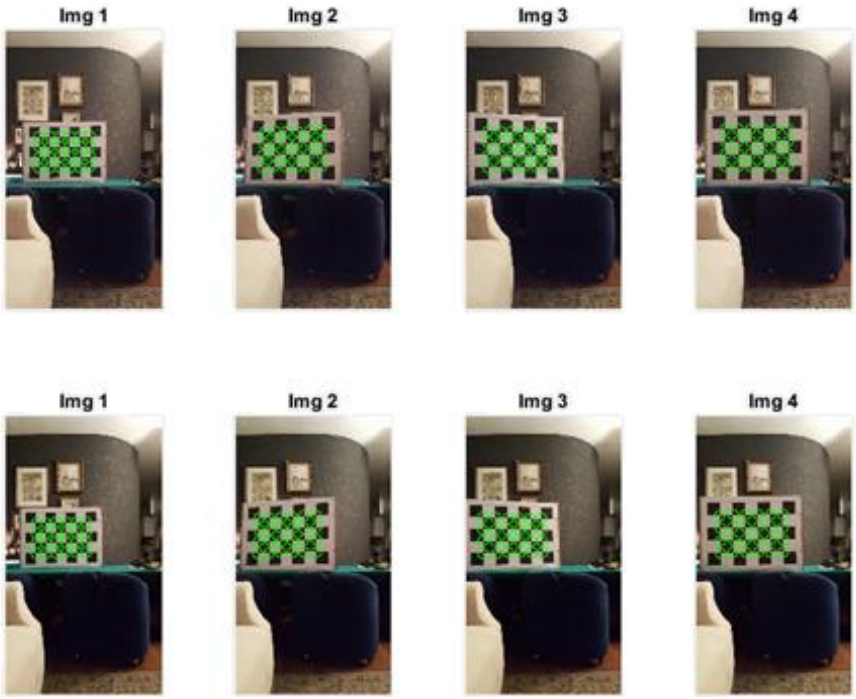


Figura 66 – Terzo test: immagini di calibrazione con Samsung e Samsung, rilevamento punti

L'errore di riproiezione medio per immagine e l'orientamento relativo delle camere ausiliare (rinominate in questa sede C_1 e C_2) sono mostrati rispettivamente in

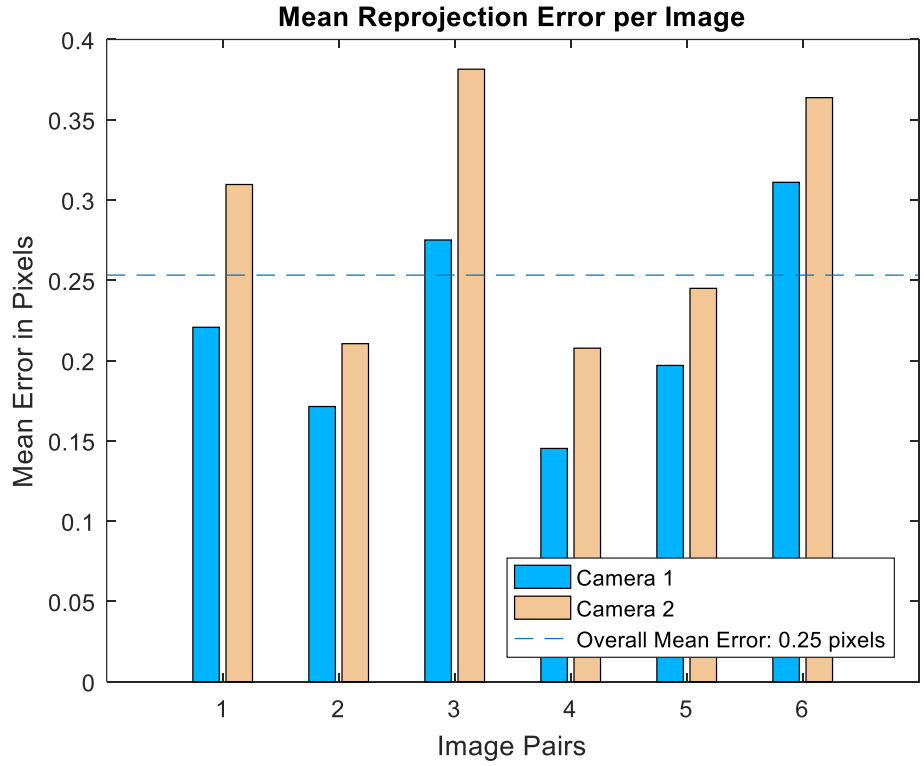


Figura 67 – Terzo test: errore di riproiezione sulle immagini di calibrazione, Samsung e Samsung

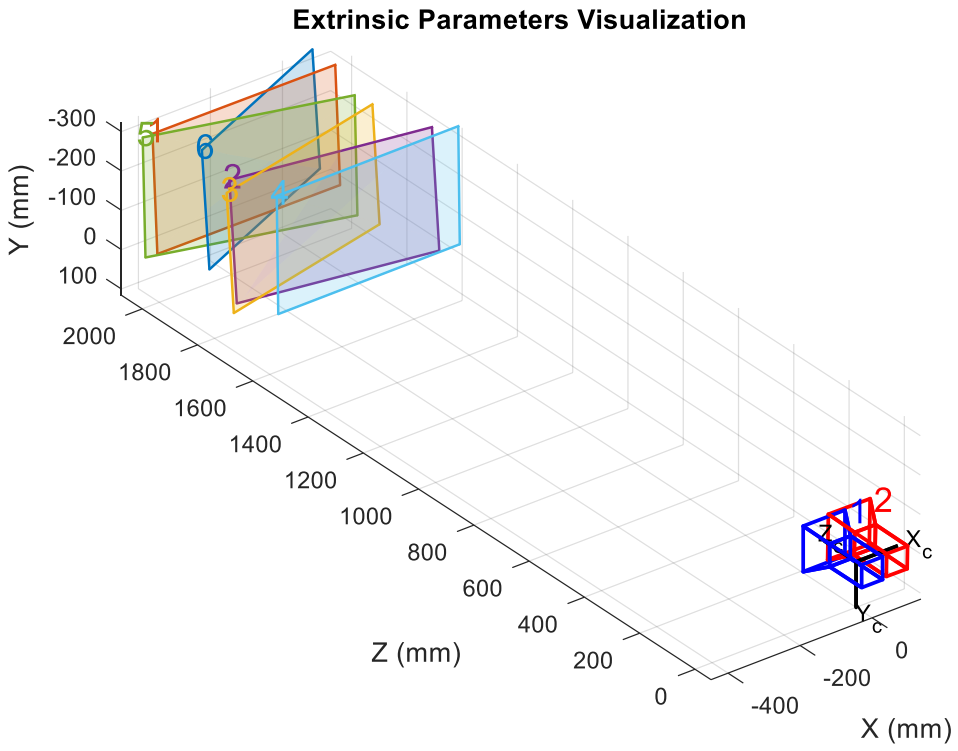


Figura 68 – Terzo test: visualizzazione settaggio delle camere, Samsung e Samsung

In questa sede, ci si vuole focalizzare sulla determinazione della scena tridimensionale. Per questa ragione non si è effettuata anche la calibrazione dell'osservatore nascosto, chiamato ora C_0. Ad esso, si suppone una semplice traslazione pari a [-.35 100 0] m rispetto all'origine, ed una rotazione di 10° attorno all'asse y della camera.

Sia C_1 la fotocamera di sinistra, e C_2 quella di destra. Il setup sperimentale è geometricamente illustrato in *Figura 69*:

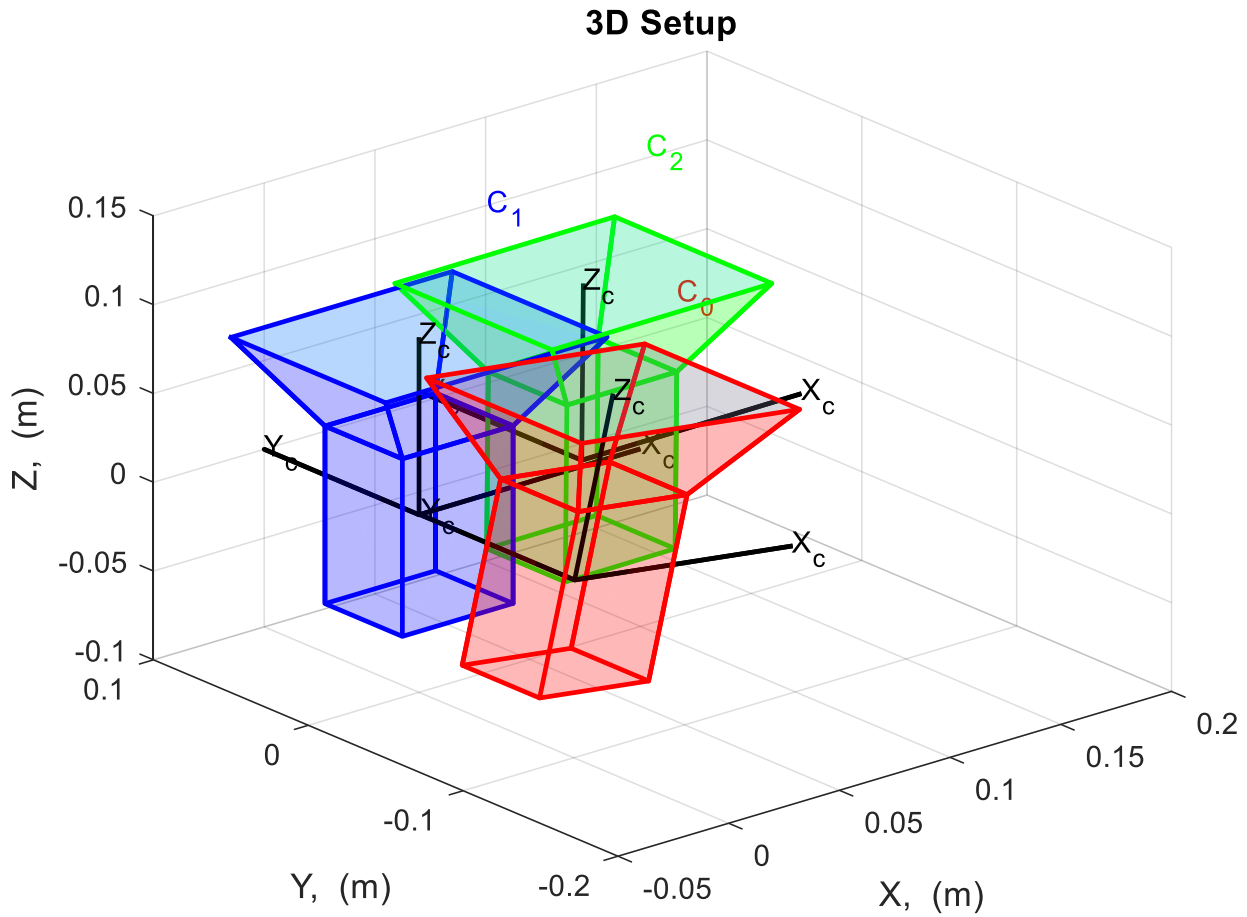
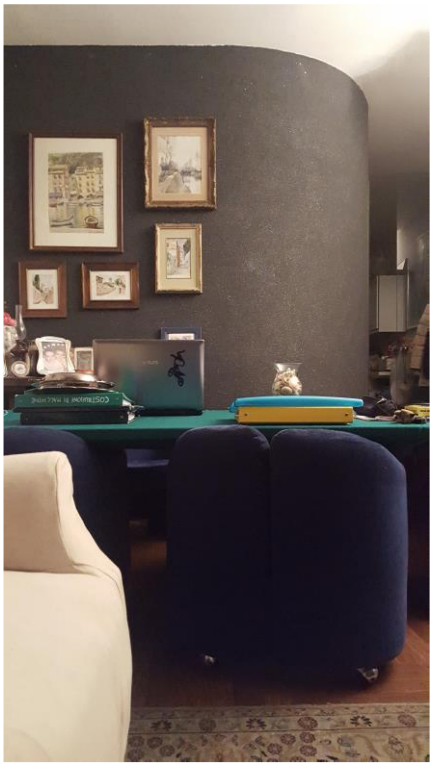


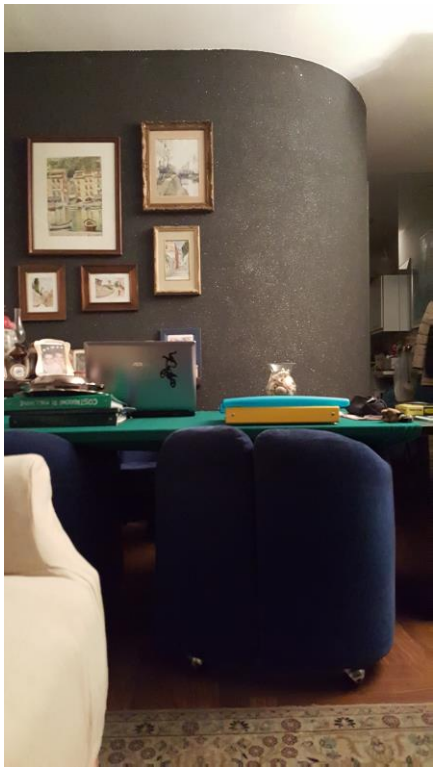
Figura 69 – Terzo test: dettaglio posizione ed orientamento camere, Samsung e Samsung

Di seguito i dettagli della coppia di immagini stereo assunta come input, e la coppia di immagini rettificata mediante la solita funzione Matlab di rettifica.

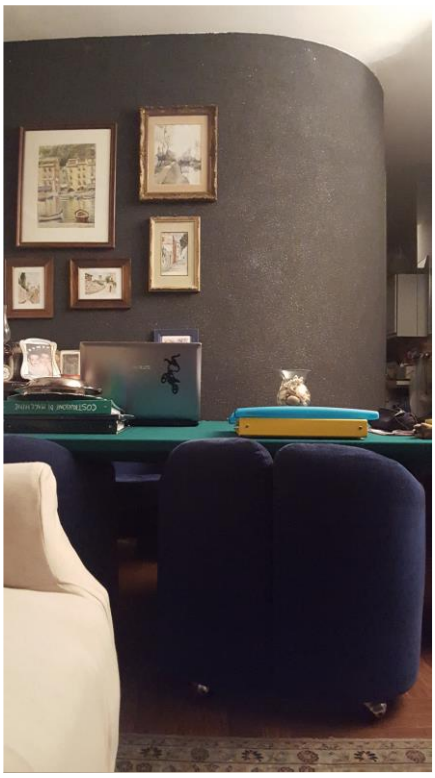
Captured scene by C_1



Captured scene by C_2



Realigned image from C_1



Realigned image from C_2

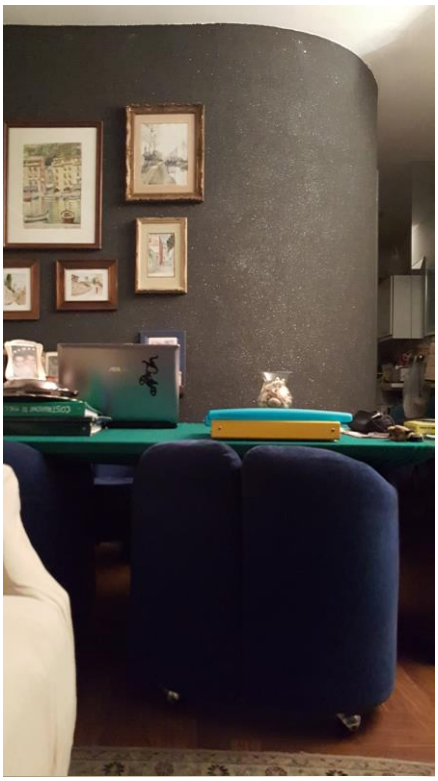


Figura 70 – Terzo test: immagini input e rettificate per generazione 3D, Samsung e Samsung

Osservazione: il processo di riallineamento delle immagini ha causato un'evidente errore nella ricostruzione dell'immagine stessa. Sembra esistere una proporzionalità tra la distorsione dell'immagine e la distanza da una linea assunta come riferimento per il reallineamento dell'immagine, probabilmente il bordo del tavolo. Ciò è apprezzabile focalizzandosi sui successivi estratti delle immagini riallineate:

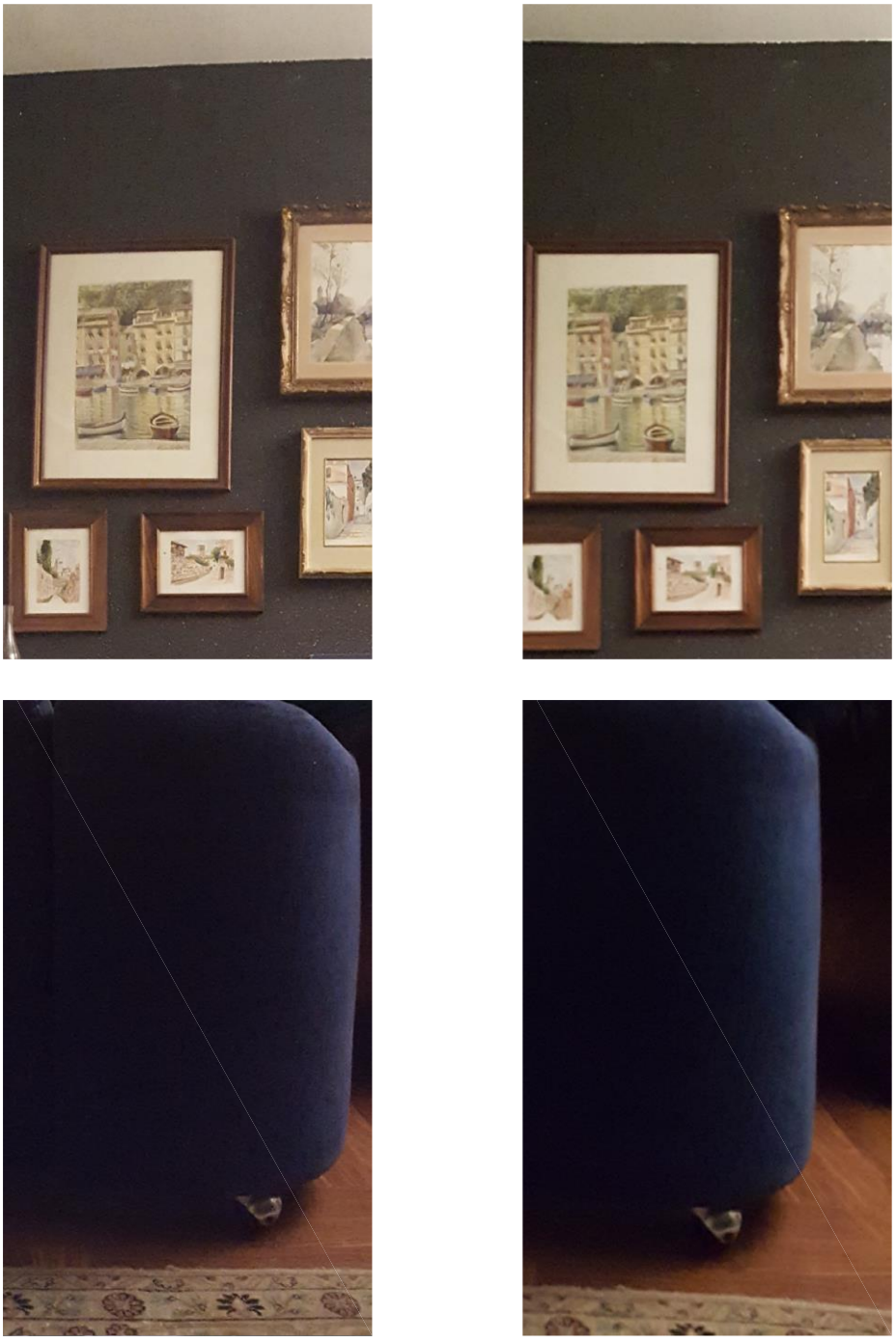


Figura 71 – Terzo test: estratto immagini pre-rettifica (sinistra) e post-rettifica (destra)

Conseguentemente, ci si aspetta una mappa delle disparità errata o con numerosi buchi in prossimità di quelle zone.

In *Tabella 3*, sono descritte le principali variabili di input che caratterizzano l'esito della mappa delle disparità.

DESCRIZIONE DELLE PRINCIPALI VARIABILI DI INPUT ALLA FUNZIONE *DISPARITY*	
BlockSize	Taglia della finestra di confronto assunta dall'algoritmo SAD, espressa in Pixels
BlockSizeVec	Taglia della finestra di confronto assunta dall'algoritmo SAD, espressa in Pixels. Utile nel seguito, se si vuole sovrapporre più mappe e colmare i buchi della mappa stessa (G*ap filling*)
DisptyRnge	[minDisparity maxDisparity], espresso in pixels. Range di ricerca delle possibili corrispondenze tra le immagini analizzate
UniqssTHold	Accrescere questo valore per imputare più pixels come inaffidabili. 15 (default)
TextrThold	Threshold minima della maglia di pixels. Abbassando questo valore, si da luogo a mappe di disparità meno affidabili. 2e-4 (default)

Tabella 3 – Descrizione delle principali variabili di input alla funzione *disparity*

In *Figura 72* si riporta la mappa delle disparità ottenuta settando i principali parametri di input ai valori riportati in *Tabella 4*:

VALORI INPUT PER OTTENERE LA MAPPA DELLE DISPARITA' DI *Figura 72*	
BlockSize	11
BlockSizeVec	/
DisptyRnge	[0 128]
UniqssTHold	8
TextrThold	2e-8

Tabella 4 – Valori input per ottenere la mappa delle disparità di *Figura 72*

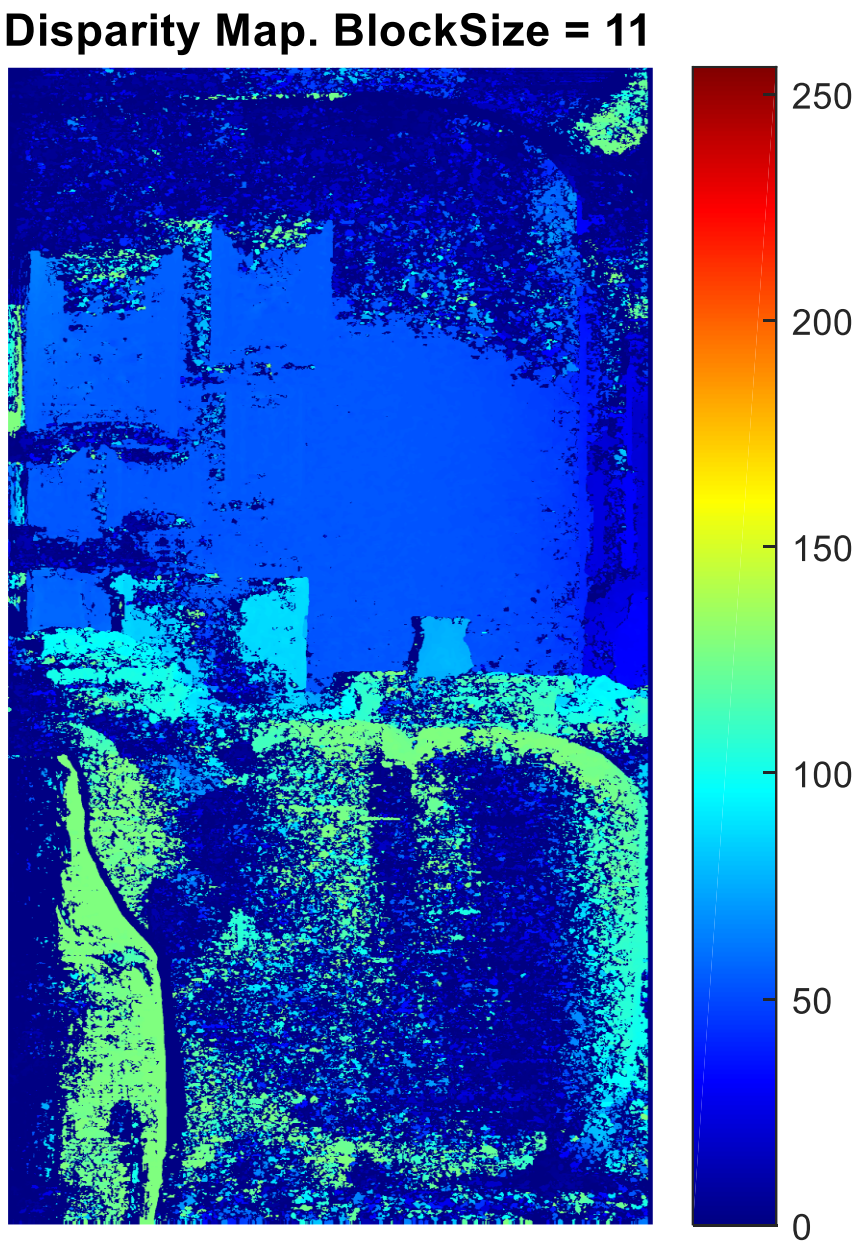


Figura 72 – Terzo test: mappa delle disparità, caso reale, Samsung e Samsung

Data la sufficiente qualità della mappa delle disparità, si opta dapprima per ricostruire la scena 3D:

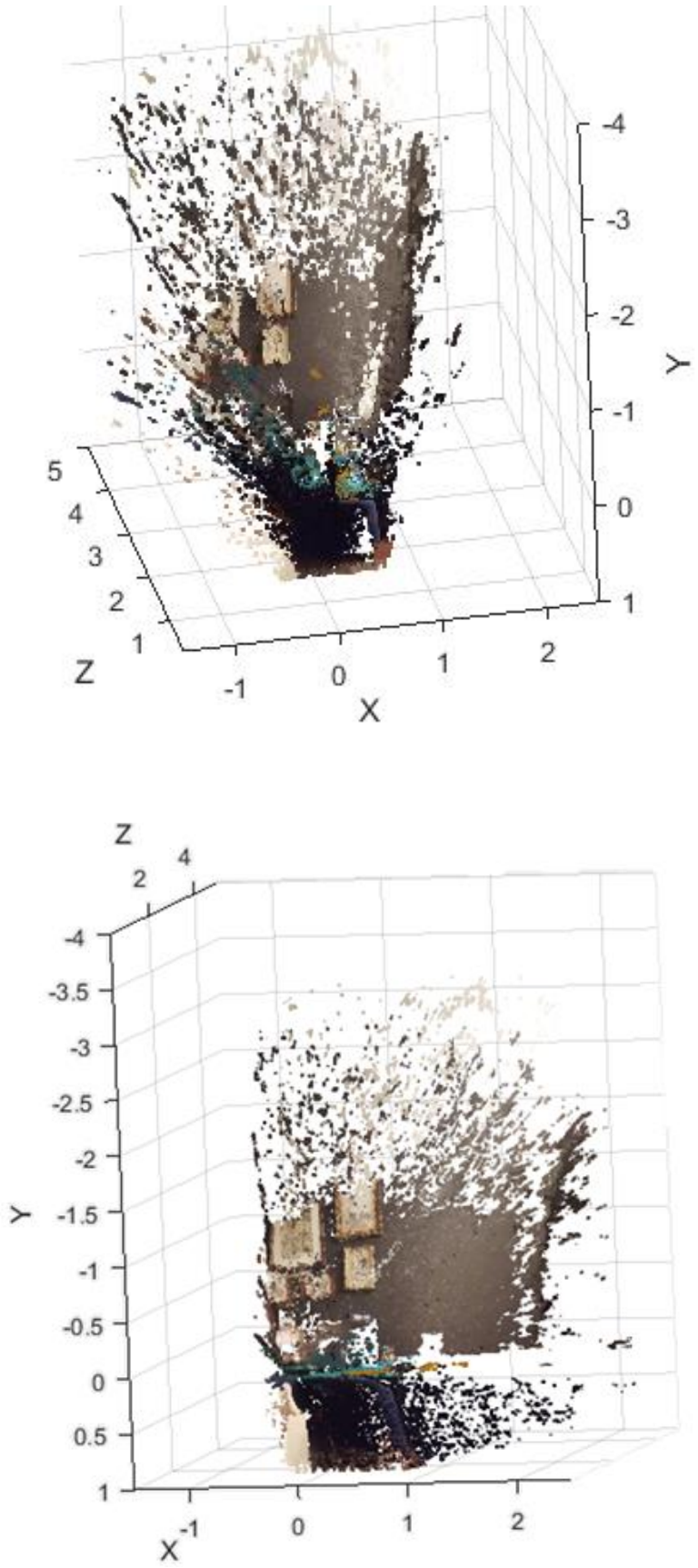


Figura 73 – Terzo test: ricostruzione scena tridimensionale (pointCloud), indoor

Di seguito si eseguirà la sintesi dell'immagine mediante l'utilizzo del modello matematico della fotocamera, descritto nel *Capitolo 3*.
L'elevata risoluzione delle immagini acquisite, fa sì che la "nuvola di punti 3D" abbia una numerosità molto elevata. Come diretta conseguenza, se si volesse riproiettare punto per punto il 3D sul piano immagine 2D dell'osservatore nascosto C_0, il calcolo e la grafica richiederebbero molto tempo. Per ovviare questo problema, si è scelto di selezionare solamente un campione di punti. Come l'intuito suggerisce, diminuendo il numero di punti selezionati per la ricostruzione, diminuisce il tempo di calcolo, di contro peggiora la qualità video. Come parametri grafici, si sono create le variabili descritte in *Tabella 5*:

DESCRIZIONE DELLE PRINCIPALI VARIABILI INFICIANTI IL PLOT DELLE IMMAGINI SINTETIZZATE	
ptsStep	Passo della nuvola di punti 3D assunta per le successive proiezioni sul piano di C_0
sctrStep	Passo dei punti assunti per la ricostruzione, accesi mediante la funzione *scatter*
scatterSize	Taglia dei punti assunti per la ricostruzione, accesi mediante la funzione *scatter*

Tabella 5 – Descrizione delle principali variabili inficianti il plot dell'immagine sintetizzata

Con la mappa delle disparità di *Figura 72* ed i parametri grafici di *Tabella 6*, si ottiene l'immagine di *Figura 74*:

PRINCIPALI INPUT PER OTTENERE L'IMMAGINE SINTETIZZATA DI *Figura 74*	
BlockSize	11
BlockSizeVec	No overlap: BlockSizeVec = BlockSize
ptsStep	1
sctrStep	5
scatterSize	2.25

Tabella 6 – Principali input per ottenere l'immagine sintetizzata di *Figura 74*

Figura 74 – Primo test sintesi immagine: indoor, Samsung e Samsung, no overlap

In *Figura 75*, come anticipato si tenta di "riempire" i buchi ricalcolando nuove mappe delle disparità al variare del parametro BlockSize e sovrapponendole. In , il setting dei principali parametri:

PRINCIPALI INPUT PER OTTENERE L'IMMAGINE SINTETIZZATA DI *Figura 75*	
BlockSize	/
BlockSizeVec	[7 11 17]
ptsStep	1
sctrStep	5
scatterSize	2.25

Tabella 7 – Principali input per ottenere l'immagine sintetizzata di *Figura 75*

Figura 75 – Primo test sintesi immagine: indoor, Samsung e Samsung, sì overlap

5.5 – Test low cost D, fotocamere stesso modello, Semi-outdoor

Si opta per eseguire un altro test sotto la luce diurna, e con ambiente più simile all'outdoor.
La realizzazione della stereocamera è la medesima del test esposto nel *Paragrafo 5.4*.

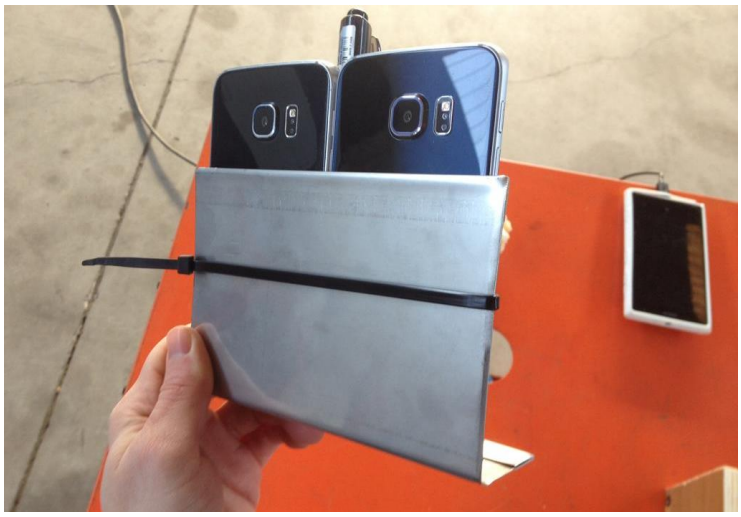


Figura 76 – Quarto setup sperimentale, camere Samsung Samsung e Nokia

Avendo maturato un minimo di esperienza per la realizzazione e manipolazione del 3D, in questo test ci si spinge sino a confrontare l'immagine sintetizzata sul piano immagine di C_0 con l'immagine realmente acquisita dalla camera C_0 (Cellulare Nokia). Per questa ragione, si rende necessaria la conoscienza della posizione, orientamento e lunghezza focale dell'osservatore C_0. Questo implica di dover calibrare anche la fotocamera dello smartphone Nokia. Il setup sperimentale allestito durante la fase di calibrazione è illustrato in *Figura 77*:

Figura 77 – Quarto setup sperimentale, fase di calibrazione, camere Samsung Samsung e Nokia

L'esito dei parametri di calibrazione estrinseci delle camere C_1, C_0 e delle camere C_1, C_2 è mostrato rispettivamente in *Figura 78* e *Figura 79*:

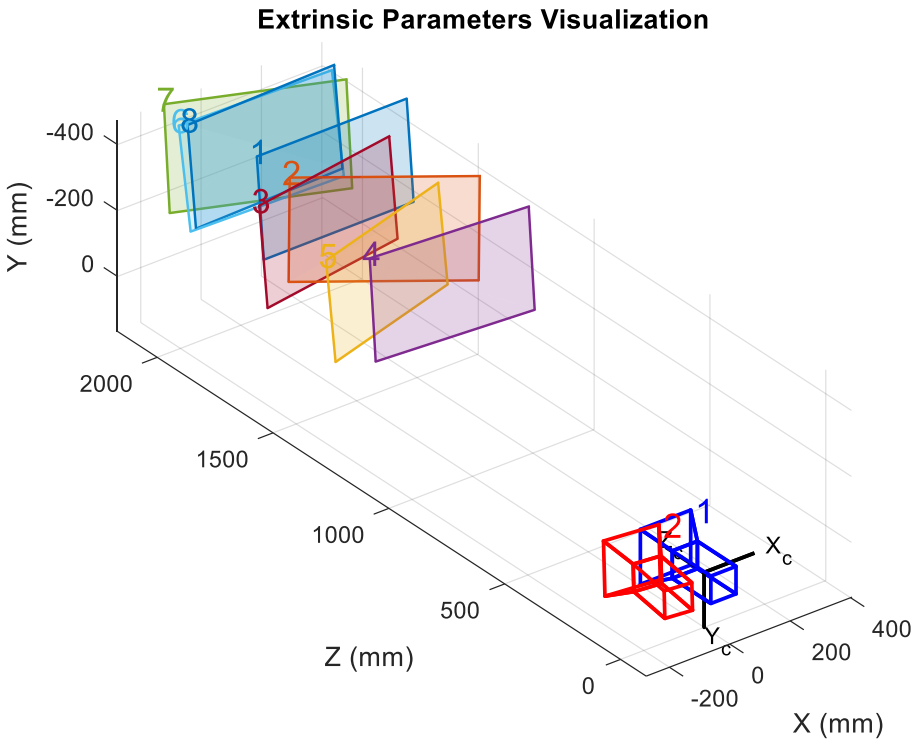


Figura 78 – Quarto test: visualizzazione settaggio delle camere, Samsung e Nokia

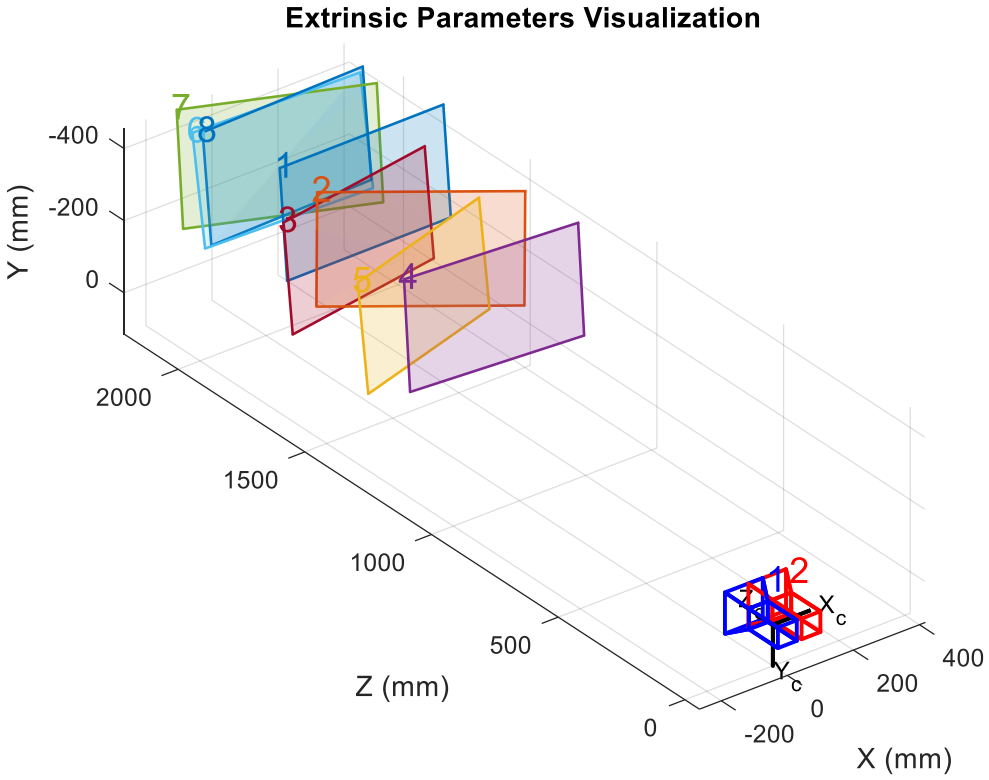


Figura 79 – Quarto test: visualizzazione settaggio delle camere, Samsung e Samsung

In *Figura 80*, l'interpretazione geometrica del risultato delle due calibrazioni eseguite pocanzi.

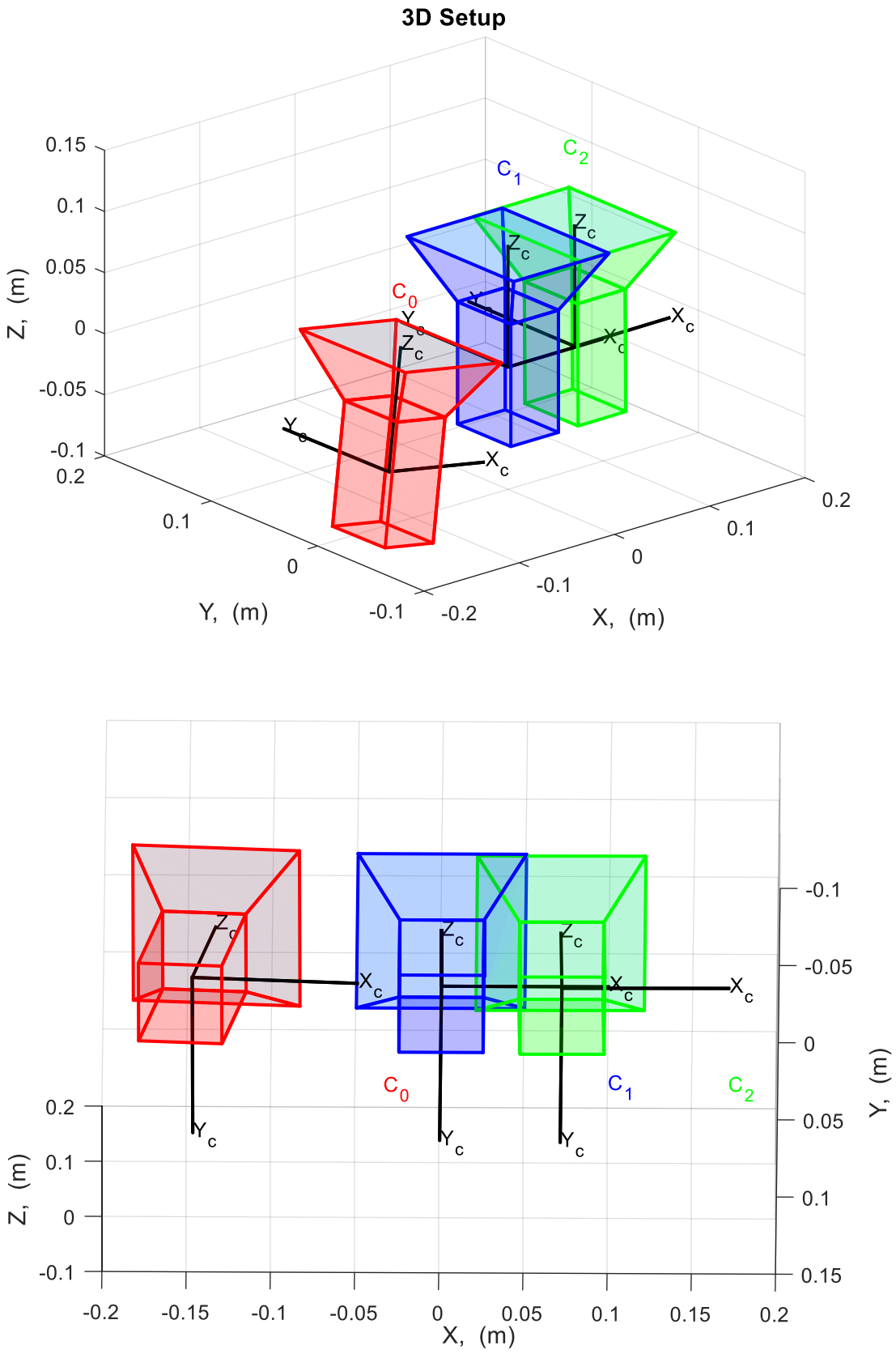


Figura 80 – Quarto test: dettaglio posizione ed orientamento camere, Samsung Samsung Nokia

In *Figura 81*, le immagini catturate dai due Samsung pre e post rettifica:

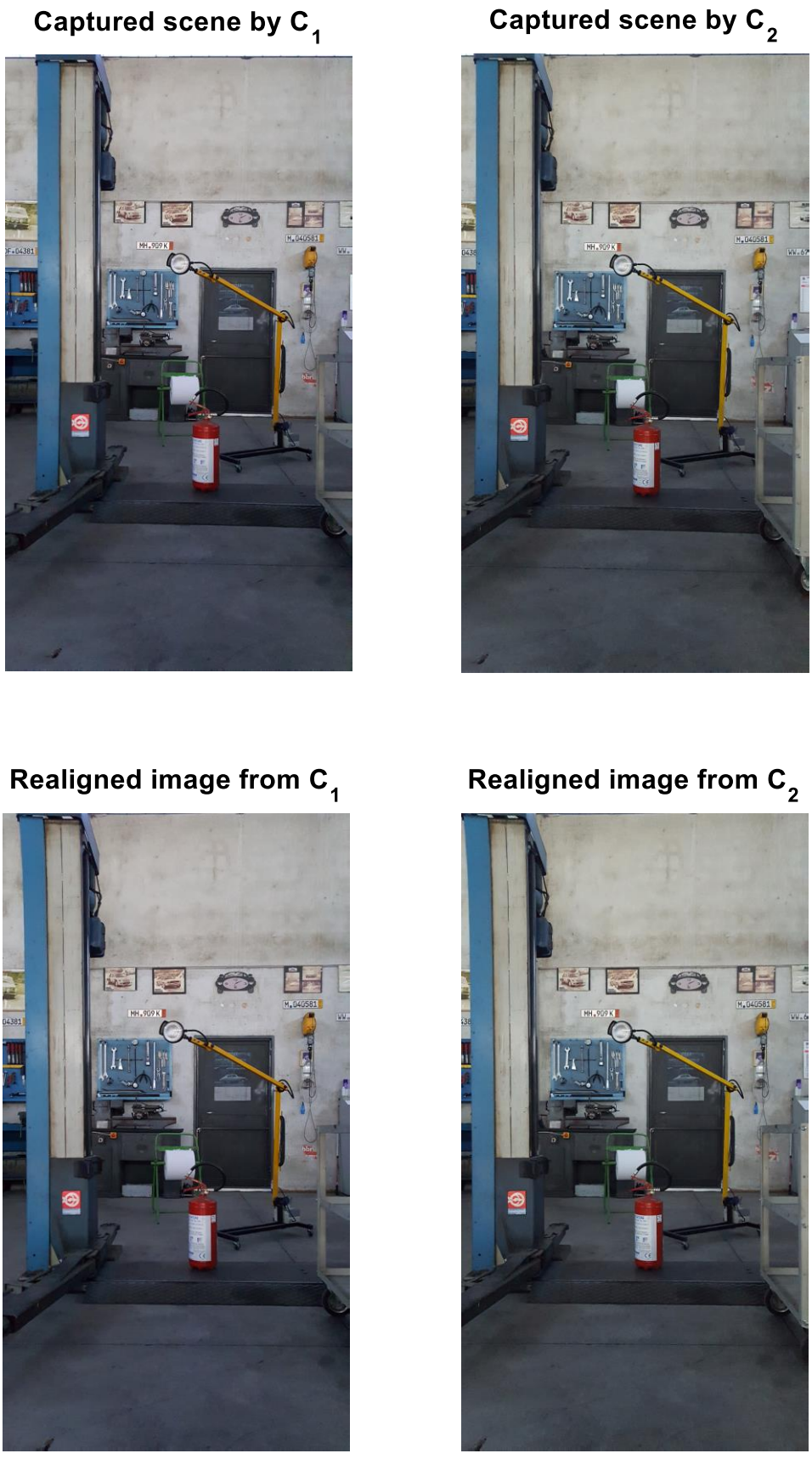


Figura 81 – Quarto test: immagini input e rettificate per generazione 3D, Samsung e Samsung

In *Figura 82*, la mappa delle disparità ottenuta con i parametri riportati in *Tabella 8*:

VALORI INPUT PER OTTENERE LA MAPPA DELLE DISPARITA' DI *Figura 82*	
BlockSize	11
BlockSizeVec	/
DisptyRnge	[0 128]
UniqssTHold	8
TextrThold	2e-8

Tabella 8 – Principali input per ottenere l'immagine sintetizzata di *Figura 82*

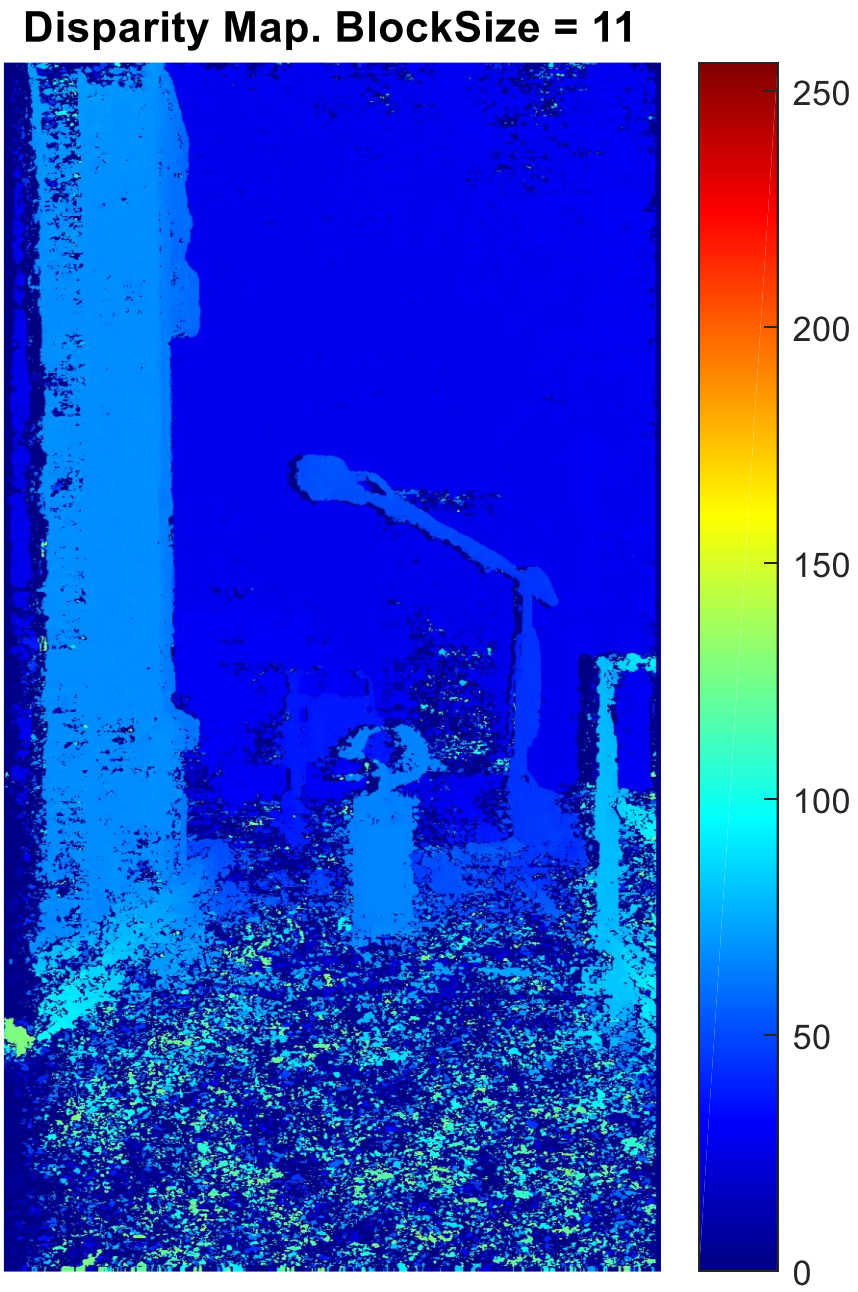


Figura 82 – Quarto test: mappa delle disparità, caso reale, Samsung e Samsung

Data la buona qualità della disparità, si opta per proseguire nel test, realizzando dapprima il 3D:

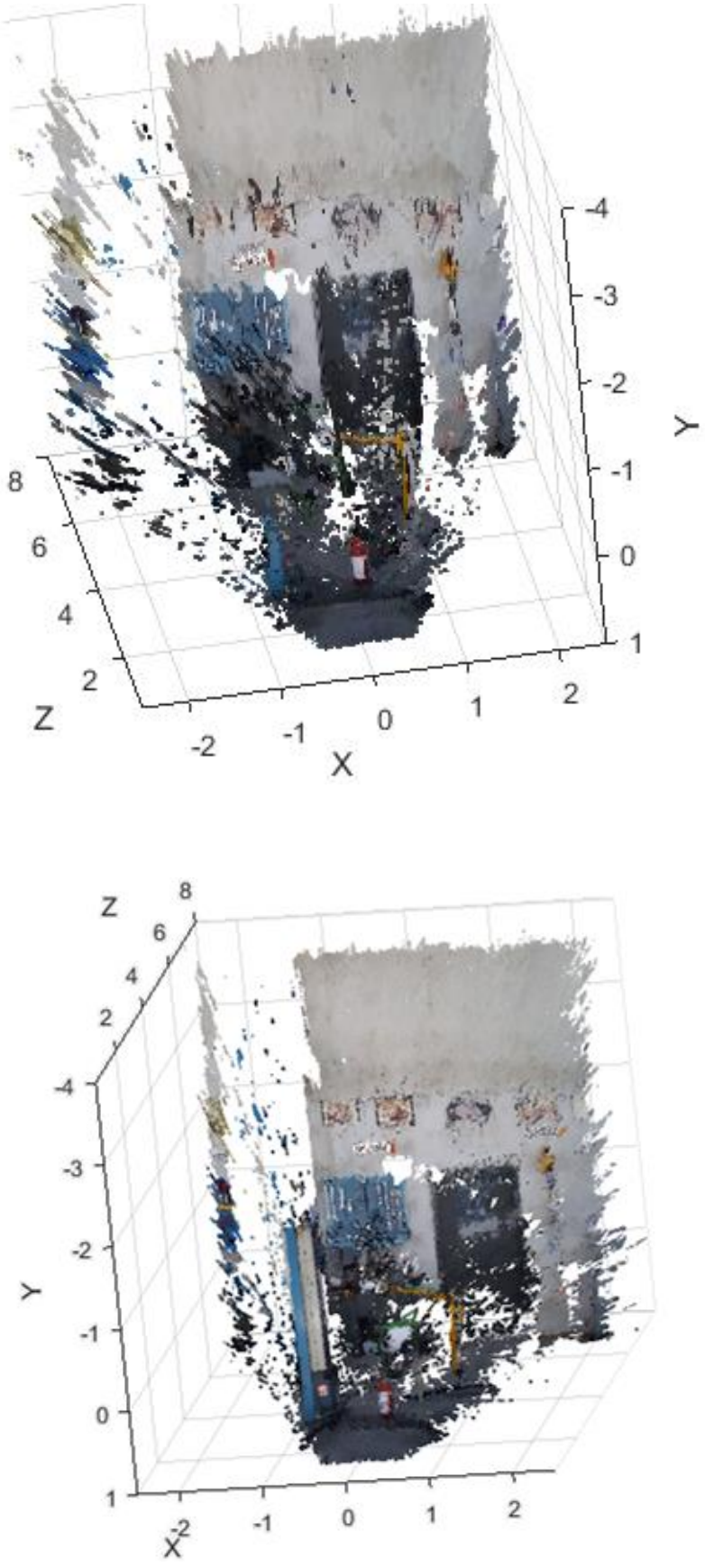


Figura 83 – Quarto test: ricostruzione scena tridimensionale (pointCloud), semi outdoor

Di seguito la sintesi dell'immagine percepita dall'osservatore C_0 ottenuta con i seguenti parametri:

PRINCIPALI INPUT PER OTTENERE L'IMMAGINE SINTETIZZATA DI *Figura 84*	
BlockSize	11
BlockSizeVec	No overlap: BlockSizeVec = BlockSize
ptsStep	1
sctrStep	10
scatterSize	2.24

Tabella 9 – Principali input per ottenere l'immagine sintetizzata di *Figura 84*

No SBD. (SBD = sorted by distance, notazione chiarita tra poco)

Figura 84 – Secondo test sintesi immagine: semi outdoor, Samsung Samsung, no overlap,no SBD

É possibile osservare che la prospettiva dell'immagine (a meno dei soliti fattori di scala della direzione orizzontale e verticale) è correttamente riprodotta. Di contro, risulta evidente che alcune zone che dovrebbero essere coperte (si faccia riferimento ad esempio alla colonna di sinistra, nell'immagine) appaiono in realtà come non nascoste e sovrastano di conseguenza i pixels associati ad una minore distanza dalla camera virtuale C_0.

É possibile risolvere il problema con un algoritmo di riordinamento, basato sulla distanza dai punti 3D dal sistema di riferimento della camera (“*sort by distance”*). In questo modo, la grafica successiva plotterà i punti dal più lontano al più vicino, “coprendo” i punti che devono risultare nascosti da oggetti “in asse” con la proiezione sul piano immagine della camera C_0.

Di seguito la sintesi della medesima immagine di *Figura 84*, ma ottenuta previa applicazione dell’algoritmo di ordinamento *SBD*.

Sì SBD.

Figura 85 – Secondo test sintesi immagine: semi outdoor, Samsung Samsung, no overlap,sì SBD

Di seguito la sintesi dell'immagine percepita dall'osservatore C_0, ottenuta con i parametri di *Tabella 10*:

PRINCIPALI INPUT PER OTTENERE L'IMMAGINE SINTETIZZATA DI *Figura 86*	
BlockSize	/
BlockSizeVec	[7 11 17]
ptsStep	1
sctrStep	10
scatterSize	2.24

Tabella 10 – Principali input per ottenere l'immagine sintetizzata di *Figura 86*

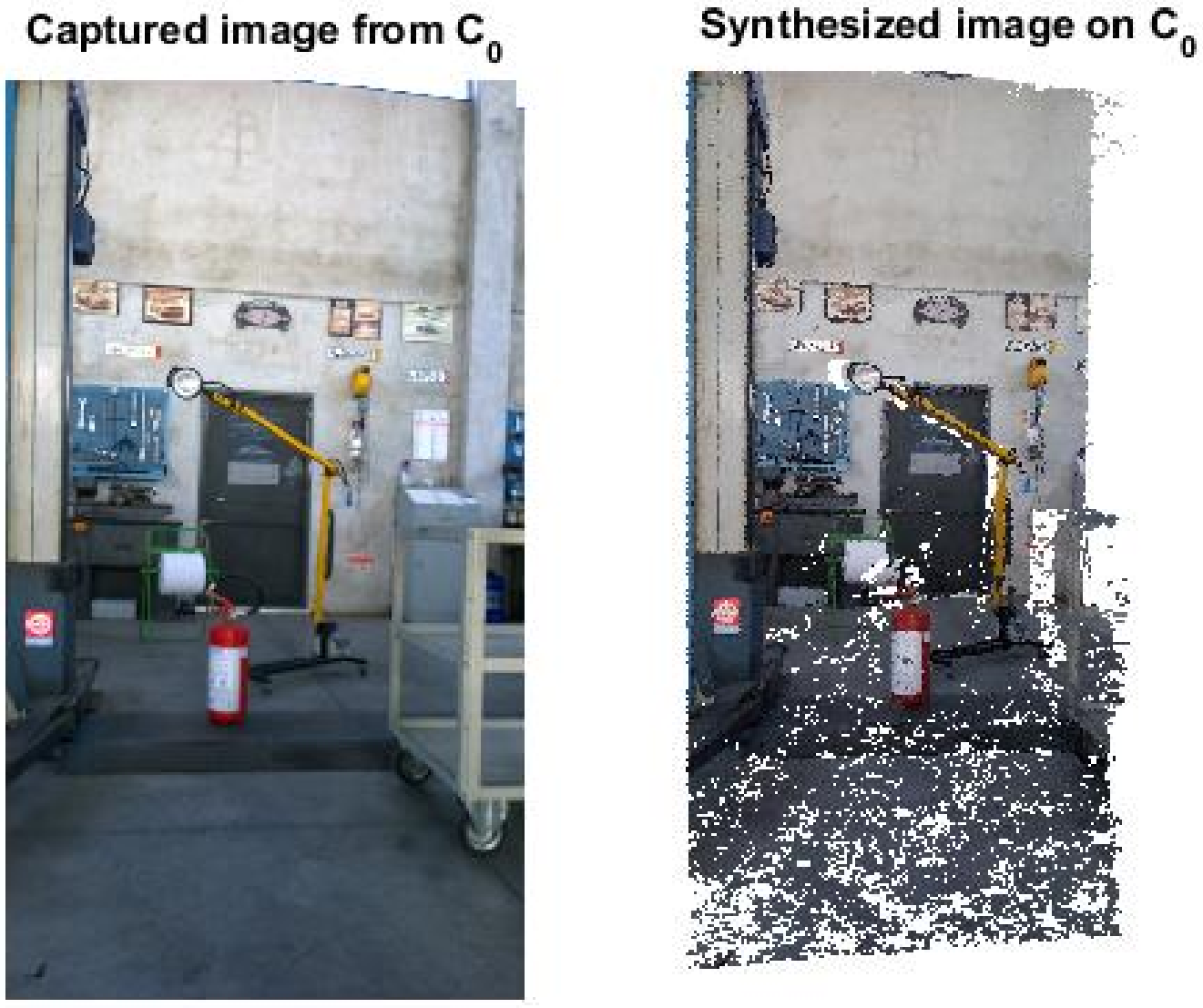


Figura 86 – Immagine reale VS sintetizzata, semi outdoor, Samsung Samsung, no overlap,sì SBD

Di seguito la sintesi dell'immagine percepita dall'osservatore C_0, ottenuta con i parametri di *Tabella 11*:

PRINCIPALI INPUT PER OTTENERE L'IMMAGINE SINTETIZZATA DI *Figura 87*	
BlockSize	/
BlockSizeVec	[7 11 15 17]
ptsStep	1
sctrStep	10
scatterSize	2.24

Tabella 11 – Principali input per ottenere l'immagine sintetizzata di *Figura 87*

Figura 87 – Secondo test sintesi immagine: semi outdoor, Samsung Samsung, sì overlap,sì SBD

6 – IMAGE SINTHESYS, REAL TIME

6.1 – Scelta delle fotocamere

Dopo aver appurato i consistenti tempi necessari alla ricostruzione dell'immagine, si decide di acquistare delle fotocamere aventi una risoluzione più bassa. In questo modo, il numero di punti oggetto di processamento si riduce, e con esso il tempo di calcolo. Di contro, la qualità delle immagini riprodotte potrebbe peggiorare, specie nel caso in cui la qualità del dispositivo sia inferiore.
In *Tabella 12*, il massimo numero di punti processabili al variare della risoluzione delle immagini:

RISOLUZIONE VS NUMERO DI PUNTI TOTALI		
NOME DEL FORMATO	PIXELS (L x H)	PIXELS TOTALI
(1) QVGA	320 x 240	76 800
(2) VGA	640 x 480	307 200
"Formato utilizzato nei test di *Capitolo 5*"	3264 x 1836	5 992 704

Tabella 12 – Risoluzione VS numero di punti totali

Dopo avere acquisito famigliarità con le parole chiave per la ricerca delle potenziali fotocamere, si esegue la ricerca principalmente nelle seguenti piattaforme:

- www.amazon.com
- www.alibaba.com
- www.ebay.com

Si è scelto di estendere la ricerca al di fuori dall'Italia in quanto il materiale trovato all'interno del dominio italiano è stato valutato come non sufficiente per una scelta consapevole.

Si tenga presente che il processo di calibrazione tenta di attribuire alla camera una "lunghezza focale equivalente" secondo un modello matematico analogo a quello di *Capitolo 3*. Per questa ragione, è di basilare importanza che la messa a fuoco dell'obiettivo sia fissa. Se si ignorasse quest'aspetto, l'algoritmo di calibrazione stimerebbe una lunghezza focale sulla base di fotografie scattate con differenti settaggi di detto parametro: si otterrebbero dati assolutamente incoerenti.

Si consideri inoltre che l'acquisto diretto di una stereo camera avrebbe condotto notevoli risparmi in termini di tempo da investire nello sviluppo software. Di contro, moto probabilmente sarebbero lievitati i costi e tale scelta non avrebbe portato ad una trattazione sufficientemente approfondita dal punto di vista teorico.

La scelta delle fotocamere si è conclusa con l'acquisto delle webcam di *Figura 88*:

Figura 88 – Webcam selezionate per la realizzazione del setup

I principali dati di acquisto sono riportati in *Tabella 13*:

PRINCIPALI DATI DI ACQUISTO DELLE WEBCAM SELEZIONATE			
NOME DISPOSITIVO	COSTO + SPESE SP	TEMPO CONSEGNA, (gg)	PROVENIENZA
MINI Spia Nascosta Pinhole VGA USB Webcam LENTE 3.7mm 640x480 60fps ad alta velocità	27.21 + 0, €	21 ÷ 39	Shenzen, Cina

Tabella 13 – Principali dati di acquisto delle Webcam selezionate

Si decide di contattare il venditore per:

- Diminuire l'elevato tempo di attesa per la spedizione
- Assicurarsi che la messa a fuoco della webcam sia fissa o impostabile come fissa
- Reperire un ordine di grandezza sul Time-delay tra l'acquisizione dell'immagine ed il successivo invio al computer
- Accertarsi sulla possibilità di collegamento diretto al pc per un'immediato utilizzo

Dati ipositivi riscontri su ciascuna domanda, si procede all'acquisto.

I principali dati inerenti al dispositivo sono riportati in *Tabella 14*.

DATASHEET DELLE WEBCAM ELP	
ELEMENTO	DETTAGLIO
Model	ELP-USB30W02M-PL37
Sensor	OV7725
Lens Size	1/4 inch
Pixel Size	6.0um X 6.0um
image area	3984 µm x 2952 µm
Max. Resolution	640(H)X480(V)
Compression format	MJPEG / YUV2（YUYV）
Resolution & frame	640X480 MJPEG@ 60fps YUY2@ 30fps/ 480X480 MJPEG@ 30fps YUY2@ 30fps 450X450 MJPEG@30fps YUY2@ 30fps/ 352X288 MJPEG@ 30fps YUY2@ 30fps 320X240 MJPEG@30fps YUY2@ 30fps/ 240X240 MJPEG@ 30fps YUY2@ 30fps
S/N Ratio	50dB
Dynamic Range	60dB
Sensitivity	3.8V/lux-sec@550nm
Mini illumination	0. 2lux
Shutter Type	Electronic rolling shutter / Frame exposure
Connecting Port type	USB2.0 High Speed
Free Drive Protocol	USB Video Class（UVC）
AEC	Support
AEB	Support
AGC	Support
Adjustable parameters	Brightness, Contrast, Saturation, Hue, Sharpness, Gamma, White balance, Backlight Contrast, Exposure
Lens Parameter	3.7mm pinhole lens
Night vision	Need to equipped IR Sensor Lens and IR LED Board
LED board power connector	Support 2P-2.0mm socket
Power supply	USB BUS POWER 4P-2.0mm socket
Power supply	DC5V
Operating Voltage	100mA~160mA
Working current	-10～70℃
Working temperature	-20～85℃
size /Weight	
Cable	Standard 1M / optional 2M,3M,5M
Operating system request	WinXP/Vista/Win7/Win8 Linux with UVC（above linux-2.6.26） MAC-OS X 10.4.8 or later Wince with UVC Android 4.0 or above with UVC

Tabella 14 – Datasheet delle webcam ELP

6.2 – Setup sperimentale

6.2.1 – Setup camere marchiate ELP

Si decide per semplicità di utilizzare 3 fotocamere dello stesso marchio e modello: inizialmente quelle marchiate ELP.

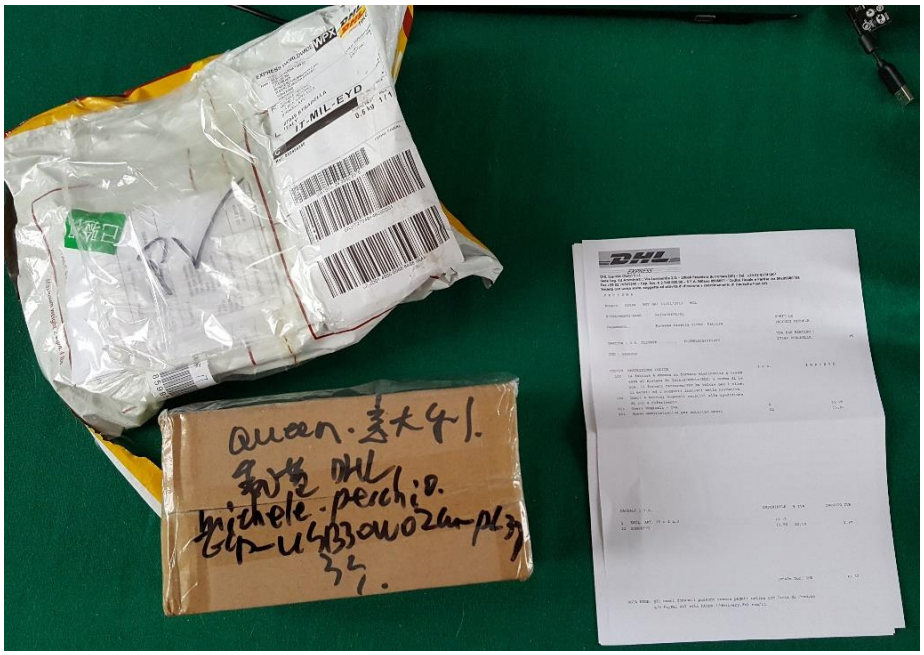


Figura 89 – Ricezione delle webcam

Sistemata la burocrazia per l'importazione, le fotocamere acquistate sono fissate su una comoda piattaforama di legno.

Figura 90 – Realizzazione del setup per webcam ELP

Per l'acquisizione della scena tridimensionale, si è scelto di posizionare due fotocamere allineate (circa) secondo il loro asse y.
In questa sede, per sempicità, le veci dell'osservatore nascosto sono fatte da un ulteriore fotocamera, collocata sulla sinistra. Tale fotocamera è stata volutamente posizionata secondo un orientamento piuttosto differente dalle altre poichè, in generale, il guidatore potrà avere i suoi occhi puntati in qualsiasi direzione.

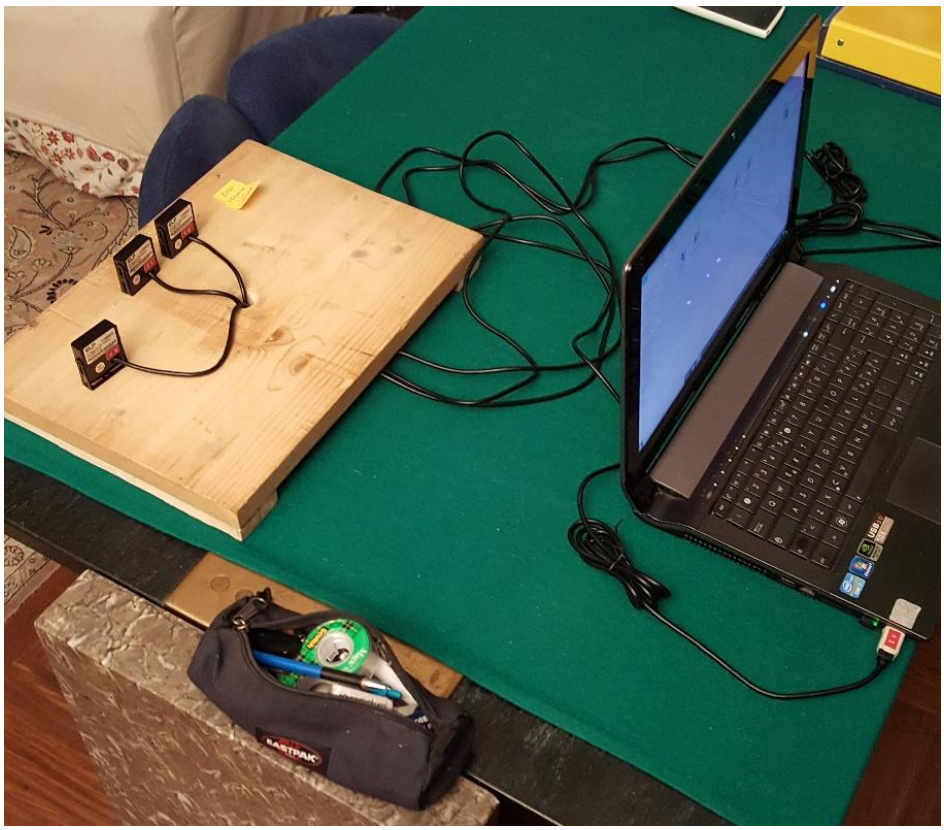


Figura 91 – Setup sperimentale, webcam ELP

Lo scopo attuale è quindi quello di confrontare l'immagine sintetizzata sulla base delle due fotocamere ausiliarie (quelle aventi asse z circa parallelo) con l'immagine realmente acquisita dall'osservatore nascosto dal montante (fotocamera C_0, sulla sinistra), in Real-time.
Si noti che in futuro, per una corretta riproduzione dell'immagine, l'osservatore C_0 dovrà essere concepito sulla base di una eyes-detection e di una stima della "lunghezza focale" degli occhi del guidatore.

6.2.1 – Setup camere marchiate Microsoft

Poiché la qualità della mappa delle disparità è risultata piuttosto scadente, sono sorti dei dubbi sulla qualità dei dispositivi utilizzati per la cattura delle immagini. Si decide pertanto di eseguire un ulteriore test con delle webcam reperite in prestito da un amico operante nel settore dell'informatica. Il nuovo setup sperimentale è dunque quello mostrato in *Figura 92*:

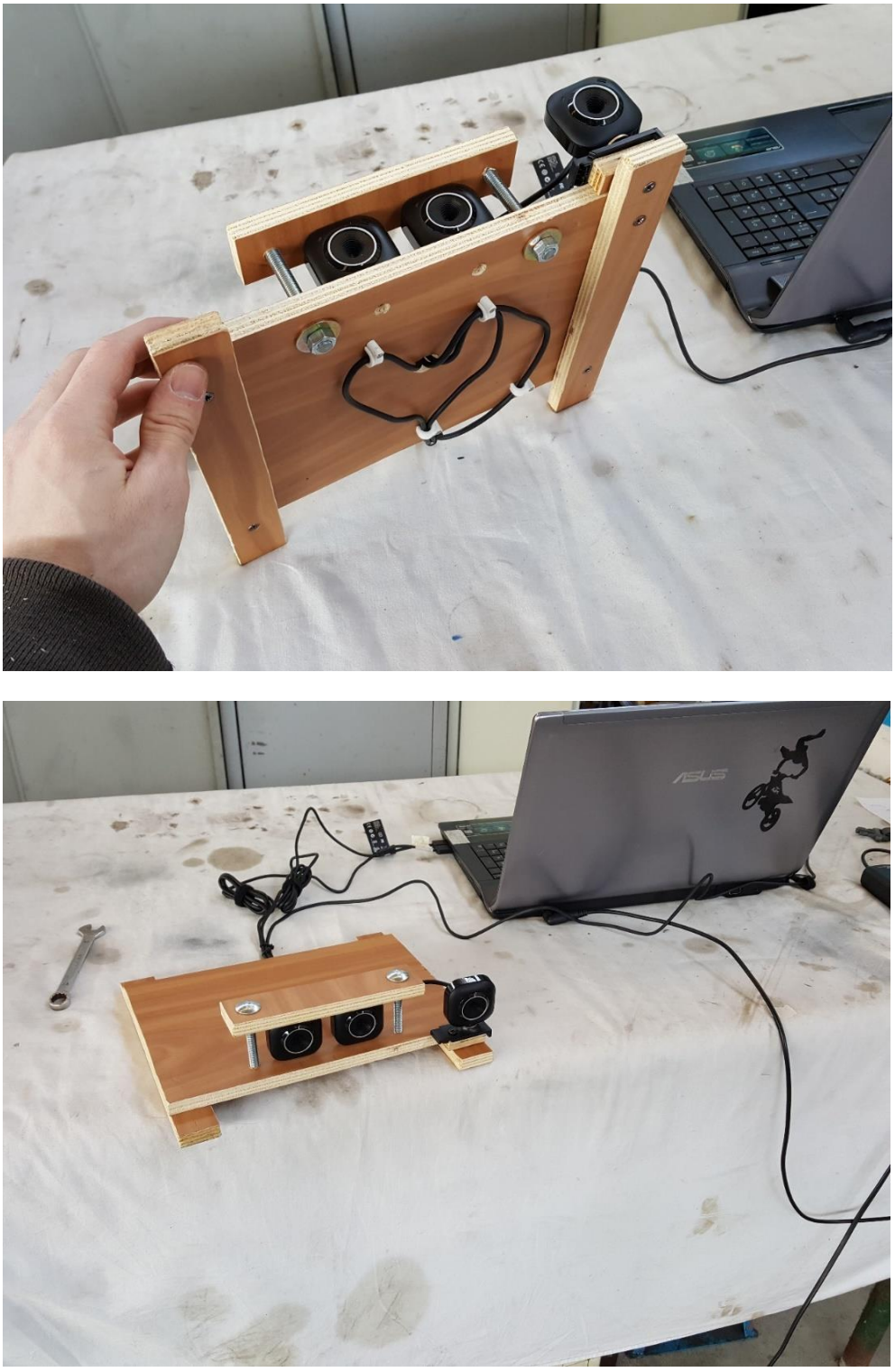


Figura 92 – Setup sperimentale, webcam Microsoft

6.3 – Logica di programmazione

6.3.1 – Acquisizione automatica delle immagini di calibrazione

Nell'ambito dell'automazione industriale si pone spesso il problema di dover effettuare la calibrazione delle fotocamere. Ciò è essenziale ogni qual volta si scelga di sostituire una camera, cambiarne posizione o semplicemente in seguito ad un urto da essa subito. Anche la semplice scelta di una differente risoluzione dello stesso dispositivo impone che si debba rieffettuare una calibrazione. Per meglio comprendere questo concetto, se a titolo di esempio si sceglio di passare da una risoluzione di 640x480 ad una risoluzione di 320x240, il field rinominato da Matlab "*FocalLength*" passa dal seguente valore:

```
>> params.CameraParameters1.FocalLength

        ans =

          731.7166  733.9533
```

A quest'ultimo valore:

```
>> params.CameraParameters1.FocalLength

        ans =

          376.8992  377.2063
```

Ed è quindi possibile affermare che tale ente viene circa dimezzato. Ciò è conseguenza diretta del fatto che, a parità di oggetto inquadrato dalla fotocamera, il numero di pixels impegnati nell'impressione dell'immagine va a dimezzarsi (in virtù della risoluzione imposta a metà del caso precedente).

Tutte queste considerazioni fanno nascere la necessità di disporre di uno script per l'acquisizione immagini sufficientemente rapido ed organizzato. Per questa ragione, è stato sviluppato l'algoritmo denominato "*ImageCapture*": esso interagisce con l'utente consentendo l'immediata visualizzazione dell'inquadratura delle 3 camere. L'utente dovrà semplicemente specificare il numero di triplette di immagini che intende scattare, e premere un qualsiasi tasto della tastiera per acquisire. La schermata che ci si trova d'innanzi è del tipo illustrata in *Figura 93*:

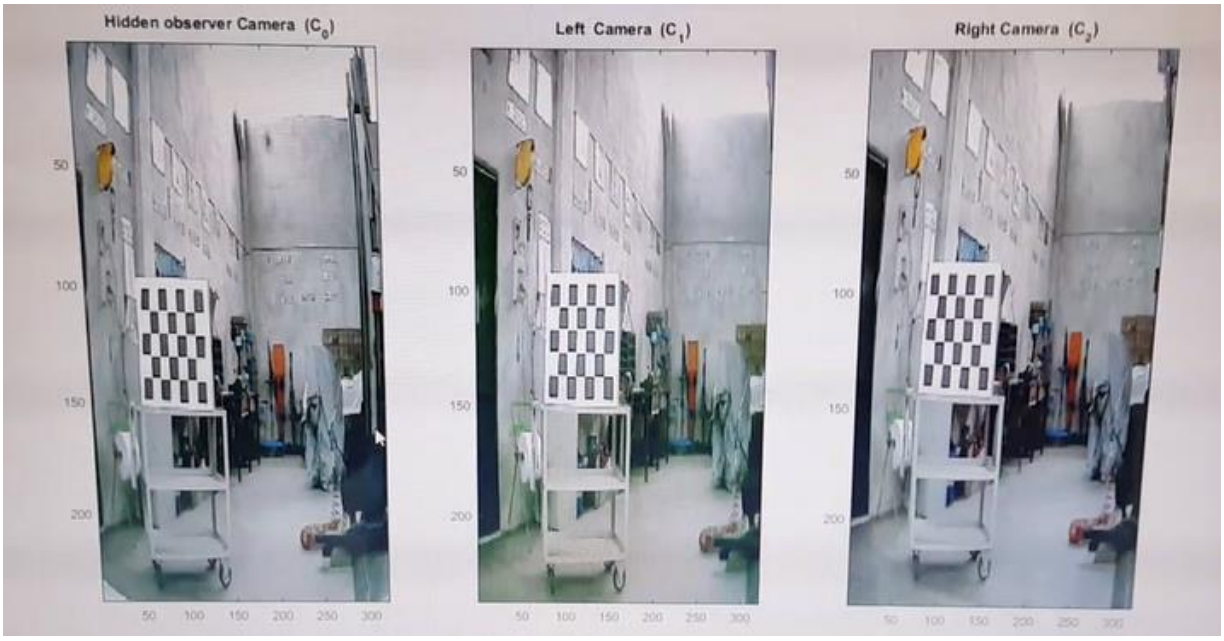


Figura 93 – Schermata algoritmo di calibrazione

Quando l'algoritmo rileva in modo corretto la CheckerBoard, è possibile acquisire la fotografia che successivamente verrà sfruttata nella procedura di calibrazione. Ci si troverà d'innanzi ad una schermata del tipo quella mostrata in *Figura 94*:

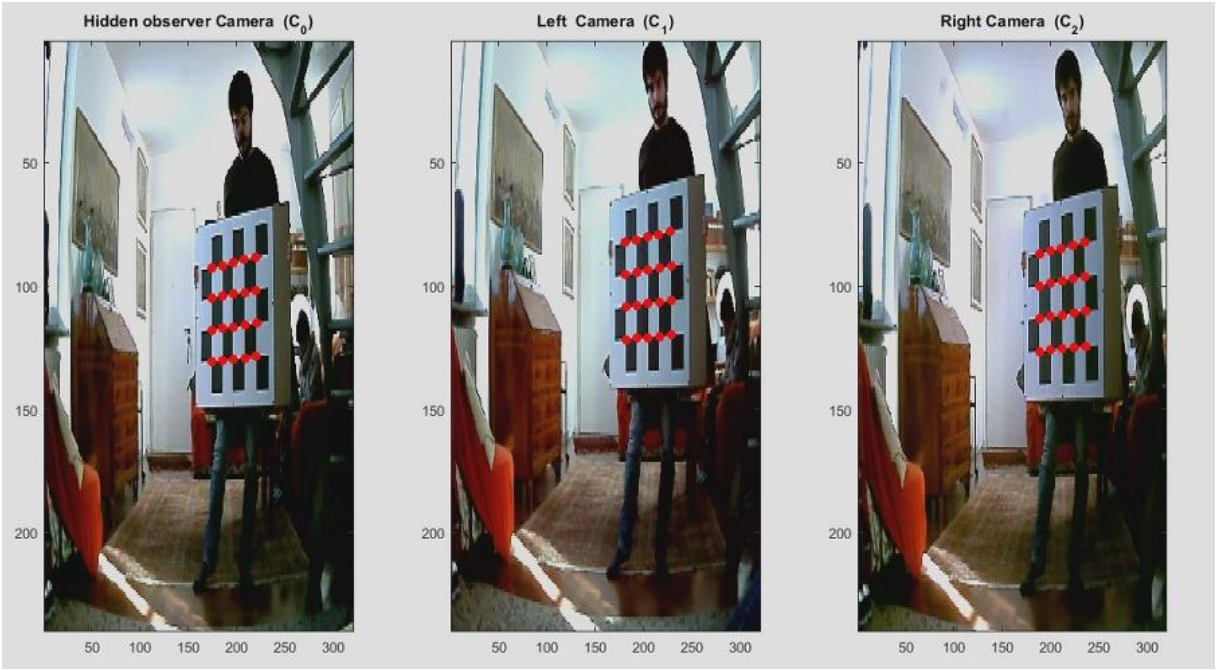


Figura 94 – Schermata algoritmo di calibrazione, rillievo CheckerBoard

Terminato di scattare le foto, il programma genererà le cartelle con le immagini opportunamente ordinate e rinominate in base alla camera da cui è stata scattata la fotografia. A titolo di esempio, si riporta quanto detto con l'illustrazione di *Figura 95*:

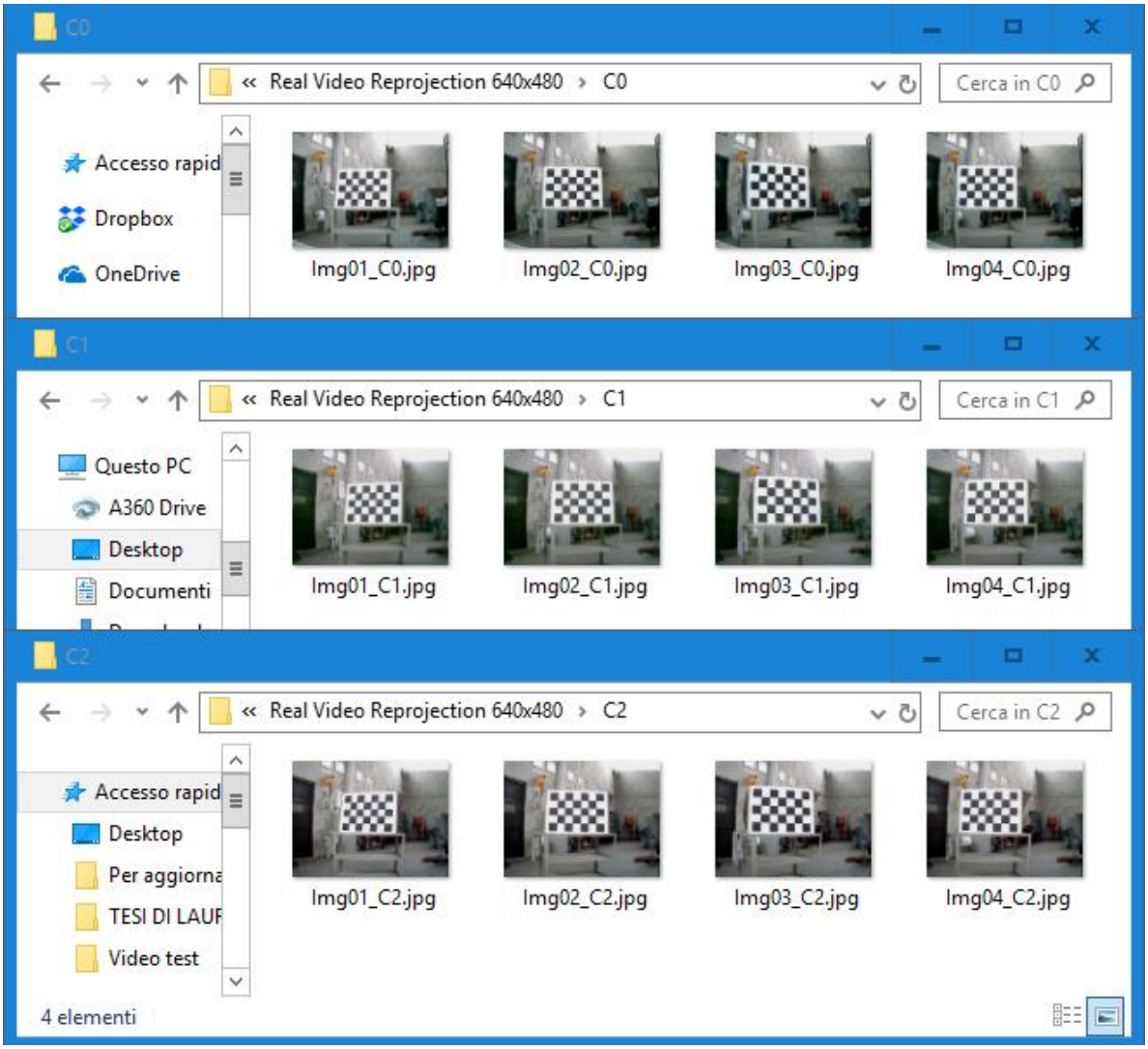


Figura 95 – Generazione automatica delle cartelle con immagini di calibrazione

In questo modo è poi possibile lanciare lo script "*Calibrazione_C1_C0*" (o "*Calibrazione_C1_C2*") che attingerà in modo automatico dalle cartelle appena generate.
Si noti che se il processo di calibrazione non dovesse andare a buon fine, talvolta è sufficiente rimuovere a mano il set di 3 immagini che causano problemi nell'algoritmo di rilevamento dei punti d'intersezione della scacchiera (o dichiarare a mano le coordinate di tali punti) e riavviare tale script.

6.3.2 – Trucchi appresi

Poiché il processo di calibrazione non sempre va a buon fine, è utile tenere in considerazione o fare riferimento alle seguenti considerazioni dedotte sperimentalmente:

- Il nome attribuito alle camere da Matlab dipende dall'ordine di inserimento delle prese USB delle stesse, e dal modello delle camere utilizzate.

- L'algoritmo di calibrazione può avere numerosi fallimenti specie per bassa risoluzione e per dimensioni del tabellone particolarmente piccole rispetto al resto dell'immagine. Ulteriori cause possono essere i riflessi e oggetti circostanti la scacchiera aventi una dimensione equiparabile a quella della scacchiera stessa.

- Le camere ELP hanno un difetto nell'obiettivo. In particolare la camera "C_0". Effettuare quindi la fotografia della ChekerBoard facendo sì che il "grumo di pixels" rovinati caschi sul nero della board e non sul bianco della stessa. Qualora si ponesse un problema analogo ma per via di sporco sulla lente, effettuare la stessa cosa.

- Se l'immagine è ricostruita in malo modo, potrebbe essere perchè le fotocamere si sono spostate e quindi scalibrate. In tal caso, rifare calibrazione.

- Se l'immagine sintetizzata non coincide bene con quella realmente catturata, potrebbe essere per via del modello pinhole non sufficientemente accurato o della lunghezza focale mal stimata. Nella realtà giocano infatti anche la distorsione delle lenti eccetera.

- Rendere gli assi z delle fotocamere ausiliarie il più paralleli possibile, ha effetti benefici sulla ricostruzione 3D. Per "allineare" il più possibile tali assi, aiutarsi inquadrando un oggetto sufficientemente lontano dagli obiettivi delle camere in questione.

- Il programma di calibrazione, se non riesce ad identificare la checkerboard, da li in poi sbaglia a plottare i punti poichè li riproietta sul subplot sbagliato. Si genera conseguentemente un errore perchè il numero di fotografie supera il numero dei set di punti rilevati. Cancellare quindi (a mano, nelle opportune cartelle) il set di 3 foto che ha causato il fallimento. Al successivo lancio del programma, i punti dovrebbero ricoincidere con la giusta foto e l'algoritmo portare a termine il suo compito di stima. Se così non fosse, rimuovere l'ulteriore set di foto e via dicendo.

- Se l'algoritmo di rilevamento dei punti non funziona con le camere ELP settate a risoluzione 640x480, può essere che sia perchè le camere sono troppo vicine alla CheckerBoard e dunque la distorsione delle lenti incide troppo sull'identificazione dei punti d'intersezione. Provare anche in modo che la distorsione dell'immagine non sia simmetrica: ad esempio scattare le foto alla Checkerboard inclinandola di molto rispetto all'asse z delle camere.
- A volte l'immagine ricostruita in Real-time "sfarfalla" un po': ciò avviene quando la grafica non riesce ad inseguire la modesta velocità del codice. Provare a diminuire la nuvola di punti da graficare o a rallentare il codice aumentando la qualità del plot.

- Se la funzione Matlab "*rectifyStereoImages*" da errore, potrebbe essere per una calibrazione di bassa qualità: scarsa luminosità dell'ambiente, numero di foto acquisite troppo basso, camere di qualità eccessivamente scarsa. Provare a settare all'interno dell'algoritmo di calibrazione i parametri:

  'EstimateSkew' -> true, 'EstimateTangentialDistortion' -> true,
  'NumRadialDistortionCoefficients' -> 2 oppure 3; e provare con tutte le varie relative combinazioni. Provare a rimuovere a mano le foto aventi errore di riproiezione maggiore.
  Controllare che il foglio su cui è stampata la ChekerBoard sia bello aderente al piano di appoggio, e assicurarsi che in qualche modo il bordo dei quadrati possa essere rilevato senza ambiguità.

- Acquisire foto alla ChekerBoard da differenti angolazioni (<45°) e distanze (Non fa male allontanarsi in modo consistente, anche sui 5 m (a seconda del tipo di camera, ovviamente)), e fare almeno 8 foto (Matlab suggerirebbe 20). Occhio ai riflessi. Posizionare la mezzeria della ChekerBoard possibilmente in prossimità dell'asse z delle fotocamere.

6.3.2 – Sintesi immagini in Real time

La sintesi dell'immagine in Real-time è semplicemente un ciclo continuo dell'algoritmo di sintesi discusso nel *Capitolo 5*. Le uniche differenze sono imputabili al diverso settaggio dei parametri grafici, al differente processo di caricamento delle immagini e allo "spegnimento" di molte delle figure di check previste nel caso di algoritmo per sintesi di immagine singola.

Da un punto di vista di schema a blocchi, l'algoritmo è quello rappresentato in *Figura 96*:

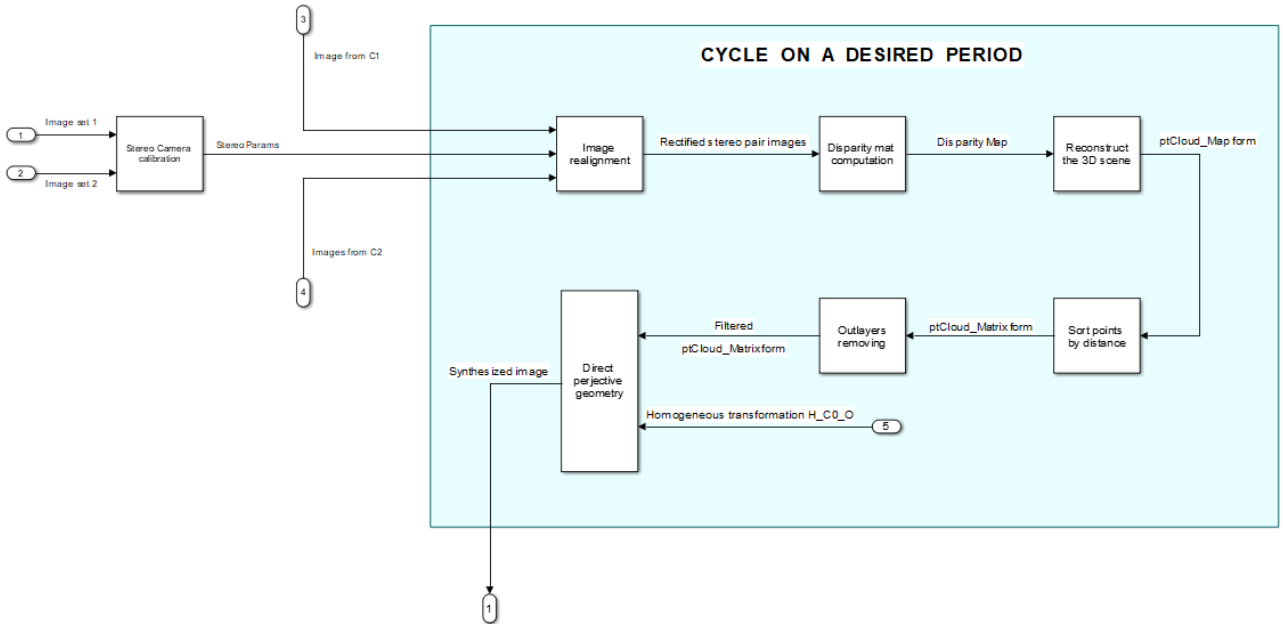


Figura 96 – Schema a blocchi per la sintesi dell'immagine, Real-time

Si tenga presente che talvolta la qualità delle immagini riportate nel seguito non è impostata al massimo possibile per la risoluzione in questione. Come per il caso di sintesi dell'immagine "statica", si è infatti scelto di graficare i punti rilevati in modo da ottenere un tempo d'esecuzione ragionevolmente basso. Tale ritardo è stato rinominato con il nome di "*Time to pixel on*", ed è fortemente dipendente dalla scheda grafica del computer utilizzato. Le principali caratteristiche del computer utilizzato per lo sviluppo della presente trattazione sono esposte in *Tabella 15*:

PRINCIPALI CARATTERISTICHE DEL COMPUTER UTILIZZATO	
Marca	Asus
Modello	N53S
Processore	Intel Core i7-270QM CPU@ 2.20GHz 2.20 GHz
Memoria Ram	8 GB
Scheda Video	Intel HD Graphics 3000, NVIDIA GeForce GT 630M

Tabella 15 – Principali caratteristiche del computer utilizzato

6.4 – Risultati sperimentali Outdoor e Semi outdoor

6.4.1 – Test con risoluzione 640x480, camere ELP

In *Tabella 16* è mostrato il settaggio dei principali parametri inficianti la velocità e qualità dell'immagine sintetizzata di *Figura 97*.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 97*		
ptsStep	sctrStep	scatterSize
1	1	5

Tabella 16 – Setting dei parametri grafici per ottenere l'immagine di *Figura 97*

L'algoritmo genera l'immagine di *Figura 97*:

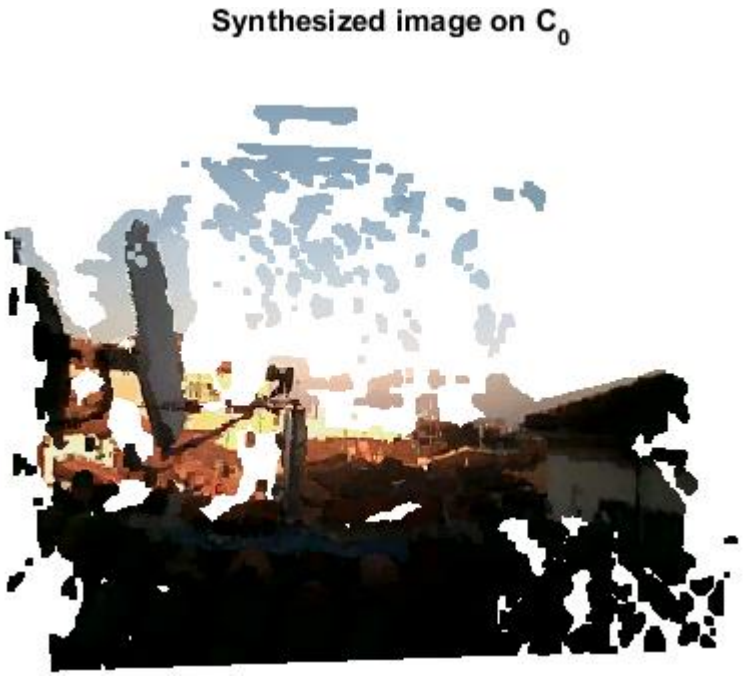


Figura 97 – Immagine sintetizzata Real-time, outdoor, camere ELP, 640x480

La qualità dell'immagine è piuttosto elevata, tuttavia la grafica del computer non riesce a rincorrere l'elevata velocità di aggiornamento dell'immagine. Si noti ancora una volta che questo è un problema puramente inerente alla grafica del computer, ciò nonostante, ritardi nell'ordine del mezzo secondo per disporre dell'immagine sono troppo elevati.

In *Tabella 17* è mostrato il settaggio dei principali parametri inficianti la velocità e qualità dell'immagine sintetizzata di *Figura 98*.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 98*		
ptsStep	sctrStep	scatterSize
4	1	25

Tabella 17 – Setting dei parametri grafici per ottenere l'immagine di *Figura 98*

L'algoritmo genera l'immagine di *Figura 98*:

Figura 98 – Immagine sintetizzata Real-time, outdoor, camere ELP, 640x480

In *Tabella 18,* un elenco delle funzioni utilizzate e del relativo dispendio temporale.

RACCOLTA TEMPI NECESSARI ALLA SINTESI DELL'IMMAGINE DI *Figura 98*	
PROCESSO	TEMPO, (s)
Images loading	0.001
Images rectification	0.011
Disparity computation	0.294
Getting world points	0.042
Reducing points density	0.002
From map form to nx6	0.003
Sorting pts by distance	0.004
Filtering outlayers	0.005
Perspective transform	0.011
Time to pixels on	ND*
TOTAL TIME NEEDED	**0.373 + ND**

Tabella 18 – Raccolta tempi necessari alla sintesi dell'immagine di ***Figura 98***

*ND = *Non Dichiarato, in quanto prevalentemente dipendente da agenti esterni al codice Matlab*

In *Figura 99* è mostrato il confronto tra l'immagine acquisita dalla camera C_0 e l'immagine sintetizzata sul piano immagine della stessa, in Real-time. Nonostante il cielo sia caratterizzato da una lunga sequenza di valori numerici simili tra di loro, la qualità è sufficientemente elevata da consentire la riproduzione parziale anche del cielo.

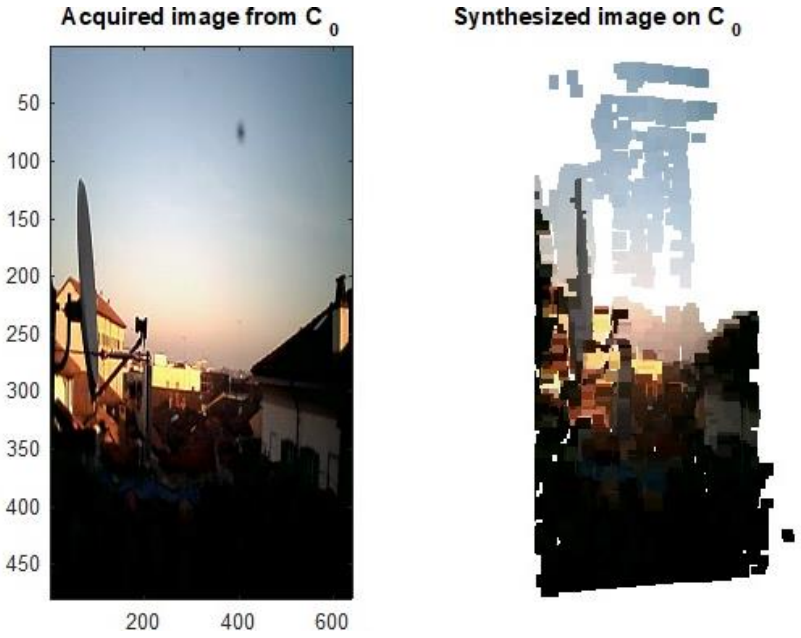


Figura 99 – Immagine sintetizzata VS acquisita, Real-time, outdoor,camere ELP,640x480

6.4.2 – Test con risoluzione 320x240, camere ELP

Il seguente test è caratterizzato da un ambiente a forte luminosità e contrasti. La bassa qualità delle camere utilizzate ha evidentemente indotto il sensore a saturare in corrispondenza dei bordi del comignolo, conseguentemente l'algoritmo di disparità non è stato in grado di ben ricostruire la scena tridimensionale. Questo spiega i grossi buchi presenti nelle immagini successive.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 100*		
ptsStep	sctrStep	scatterSize
1	1	25

Tabella 19 – Setting dei parametri grafici per ottenere l'immagine di *Figura 100*

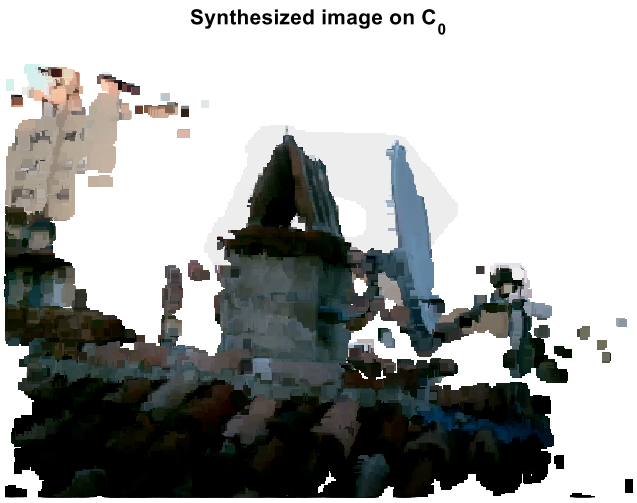


Figura 100 – Immagine sintetizzata Real-time, outdoor, camere ELP, 320x240, ptsStep 1

RACCOLTA TEMPI NECESSARI ALLA SINTESI DELL'IMMAGINE DI *Figura 100*	
PROCESSO	TEMPO, (s)
Images loading	0.003
Images rectification	0.015
Disparity computation	0.064
Getting world points	0.008
Reducing points density	0.003
From map form to nx6	0.009
Sorting pts by distance	0.010
Filtering outlayers	0.001
Perspective transform	0.006
Time to pixels on	ND
TOTAL TIME NEEDED	**0.120 + ND**

Tabella 20 – Raccolta tempi necessari alla sintesi dell'immagine di *Figura 100*

Il seguente test vede diminuire la qualità dell'immagine per disporre di un'immagine maggiormente prossima ad un Real-time vero e proprio. Come già detto, essa è una necessità dettata unicamente dalla scheda grafica del computer relativamente lenta.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 101*		
ptsStep	sctrStep	scatterSize
4	1	45

Tabella 21 – Setting dei parametri grafici per ottenere l'immagine di *Figura 101*

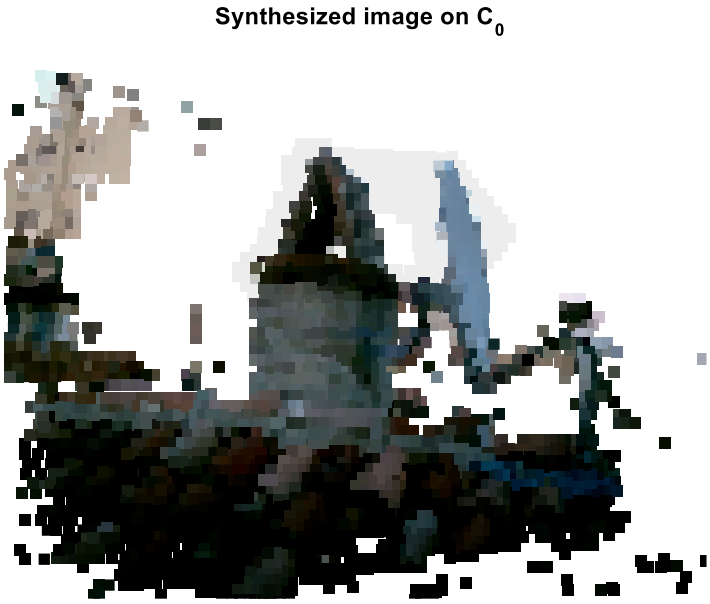


Figura 101 – Immagine sintetizzata Real-time, outdoor, camere ELP, 320x240, ptsStep 4

RACCOLTA TEMPI NECESSARI ALLA SINTESI DELL'IMMAGINE DI *Figura 101*	
PROCESSO	TEMPO, (s)
Images loading	0.000
Images rectification	0.004
Disparity computation	0.066
Getting world points	0.011
Reducing points density	0.002
From map form to nx6	0.002
Sorting pts by distance	0.001
Filtering outlayers	0.000
Perspective transform	0.001
Time to pixels on	ND
TOTAL TIME NEEDED	**0.087 + ND**

Tabella 22 – Raccolta tempi necessari alla sintesi dell'immagine di *Figura 101*

Il seguente test mira ad avere un riscontro diretto tra l'immagine acquisita e quella sintetizzata, per questa ragione, in questa sede non si presterà particolarmente attenzione al consumo di tempo. Ciò corrisponde alla scelta (a parità di tutti gli altri fattori) di un parametro *ptsStep* particolarmente basso.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 102*		
ptsStep	sctrStep	scatterSize
1	1	25

Tabella 23 – Setting dei parametri grafici per ottenere l'immagine di *Figura 102*

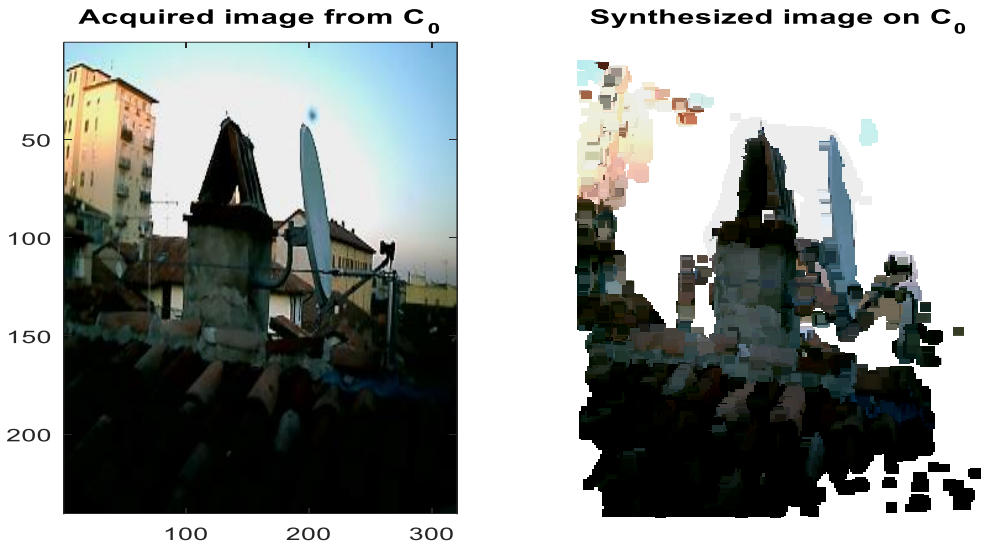


Figura 102 – Immagine sintetizzata VS acquisita, Real-time, outdoor,camere ELP,320x240

RACCOLTA TEMPI NECESSARI ALLA SINTESI DELL'IMMAGINE DI *Figura 102*	
PROCESSO	TEMPO, (s)
Images loading	0.002
Images rectification	0.009
Disparity computation	0.100
Getting world points	0.009
Reducing points density	0.007
From map form to nx6	0.013
Sorting pts by distance	0.015
Filtering outlayers	0.001
Perspective transform	0.008
Time to pixels on	ND
TOTAL TIME NEEDED	**0.164 + ND**

Tabella 24 – Raccolta tempi necessari alla sintesi dell'immagine di *Figura 102*

6.4.3 – Test con risoluzione 640x480, camere Microsoft

Poiché il setup sperimentale è in tutto e per tutto equiparabile a quello del caso precedente, non si riporterà ora l'estratto dei tempi. Ciò viene fatto a beneficio del risparmio di carta e tempo: l'ordine di grandezza nonché i relativi valori numerici sono circa identici ai vecchi. Ciò che cambia, è la qualità delle fotocamere che ora è decisamente più alta e dunque la nuvola di punti ricostruita è molto più definita e piena. Di conseguenza, l'immagine sintetizzata ha una qualità più alta.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 103*		
ptsStep	sctrStep	scatterSize
1	1	15

Tabella 25 – Setting dei parametri grafici per ottenere l'immagine di *Figura 103*

Con i valori riportati in *Tabella 25*, si ottiene l'immagine sintetizzata di *Figura 103*:

Figura 103 – Immagine sintetizzata Real-time, semi outdoor, camere Mic, 640x480,ptsStep 1

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 104*		
ptsStep	sctrStep	scatterSize
4	1	20

Tabella 26 – Setting dei parametri grafici per ottenere l'immagine di *Figura 104*

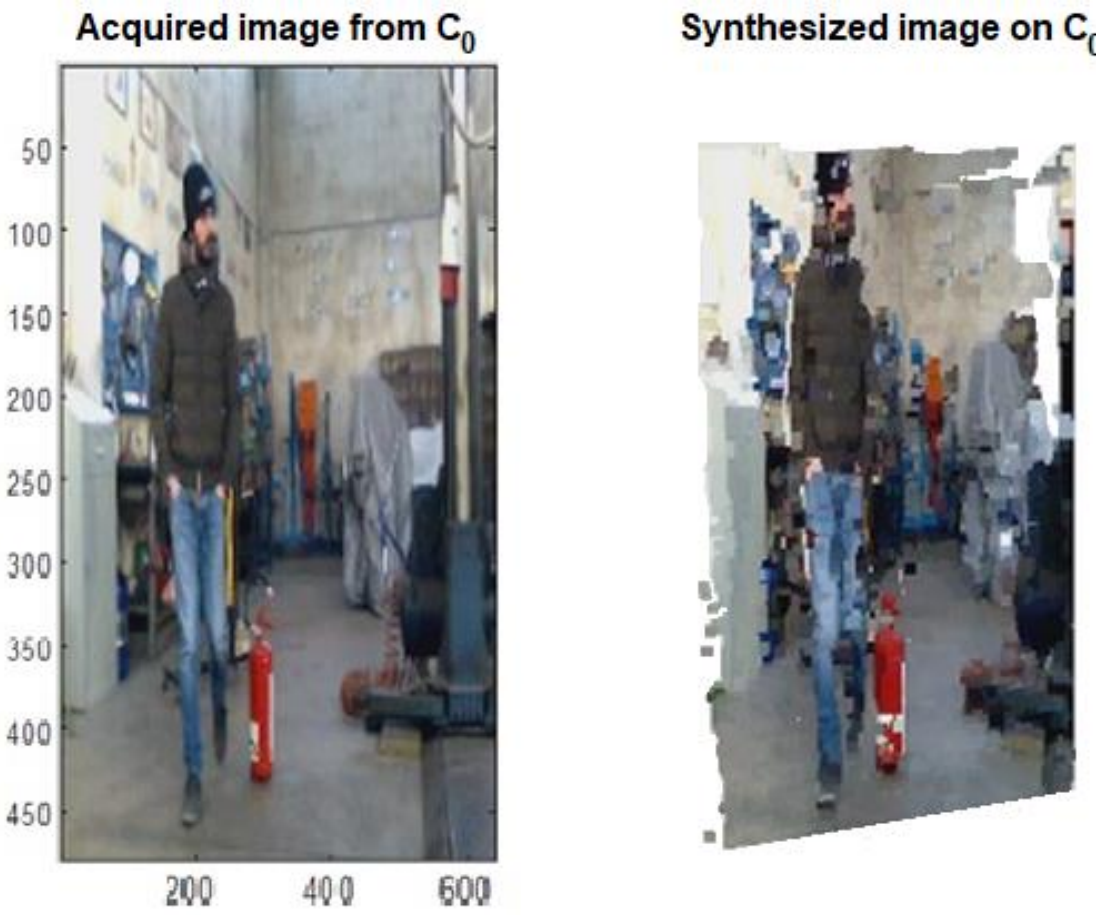


Figura 104 – Immagine sintetizzata VS acquisita, Real-time, semi outdoor,camere Mic,640x480

6.4.4 – Test con risoluzione 320x240, camere Microsoft

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 105*		
ptsStep	sctrStep	scatterSize
1	1	35

Tabella 27 – Setting dei parametri grafici per ottenere l'immagine di *Figura 105*

Figura 105 – Immagine sintetizzata Real-time, semi outdoor, camere Mic, 320x240, ptsStep 1

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 106*		
ptsStep	sctrStep	scatterSize
4	1	45

Tabella 28 – Setting dei parametri grafici per ottenere l'immagine di *Figura 106*

Figura 106 – Immagine sintetizzata VS acquisita, Real-time, semi outdoor,camere Mic,320x240

6.5 – Risultati sperimentali Indoor

6.5.1 – Test con risoluzione 640x480, camere ELP

L'ordine di grandezza dei tempi necessari alla ricostruzione è coincidente con quello del caso "outdor" e, pertanto, non verranno riportate ulteriori tabelle descrittive dei tempi necessari poichè ritenute non particolarmente significative.

La *Tabella 29* mostra il settaggio dei principali parametri inficianti la velocità e qualità dell'immagine sintetizzata di figura successiva.

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 107*		
ptsStep	sctrStep	scatterSize
1	1	5

Tabella 29 – Setting dei parametri grafici per ottenere l'immagine di *Figura 107*

L'algoritmo genera l'immagine riportata in *Figura 107*:

Figura 107 – Immagine sintetizzata Real-time, indoor, camere ELP, 640x480, ptsStep 1

6.5.2 – Test con risoluzione 320x240, camere ELP

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 108*		
ptsStep	sctrStep	scatterSize
4	1	35

Tabella 30 – Setting dei parametri grafici per ottenere l'immagine di *Figura 108*

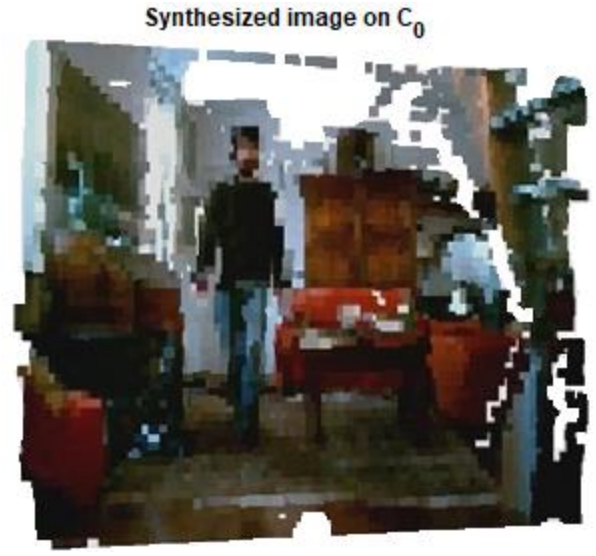


Figura 108 – Immagine sintetizzata Real-time, indoor, camere ELP, 320x240, ptsStep 4

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 109*		
ptsStep	sctrStep	scatterSize
3	1	15

Tabella 31 – Setting dei parametri grafici per ottenere l'immagine di *Figura 109*

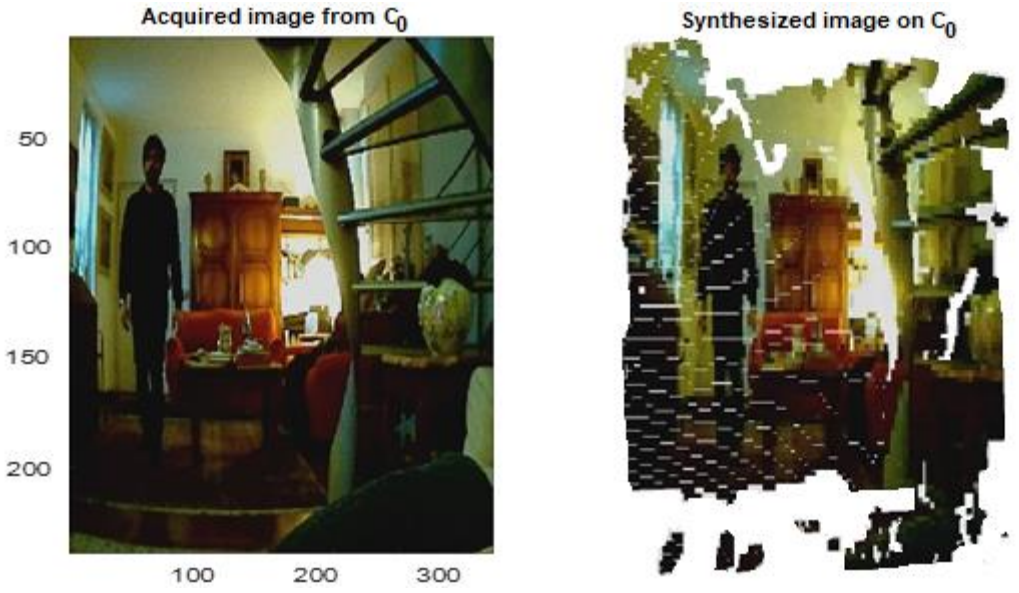


Figura 109 – Immagine sintetizzata VS acquisita, Real-time, indoor, camere ELP, 320x240

6.5.3 – Test con risoluzione 640x480, camere Microsoft

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 110*		
ptsStep	sctrStep	scatterSize
2	1	5

Tabella 32 – Setting dei parametri grafici per ottenere l'immagine di *Figura 110*

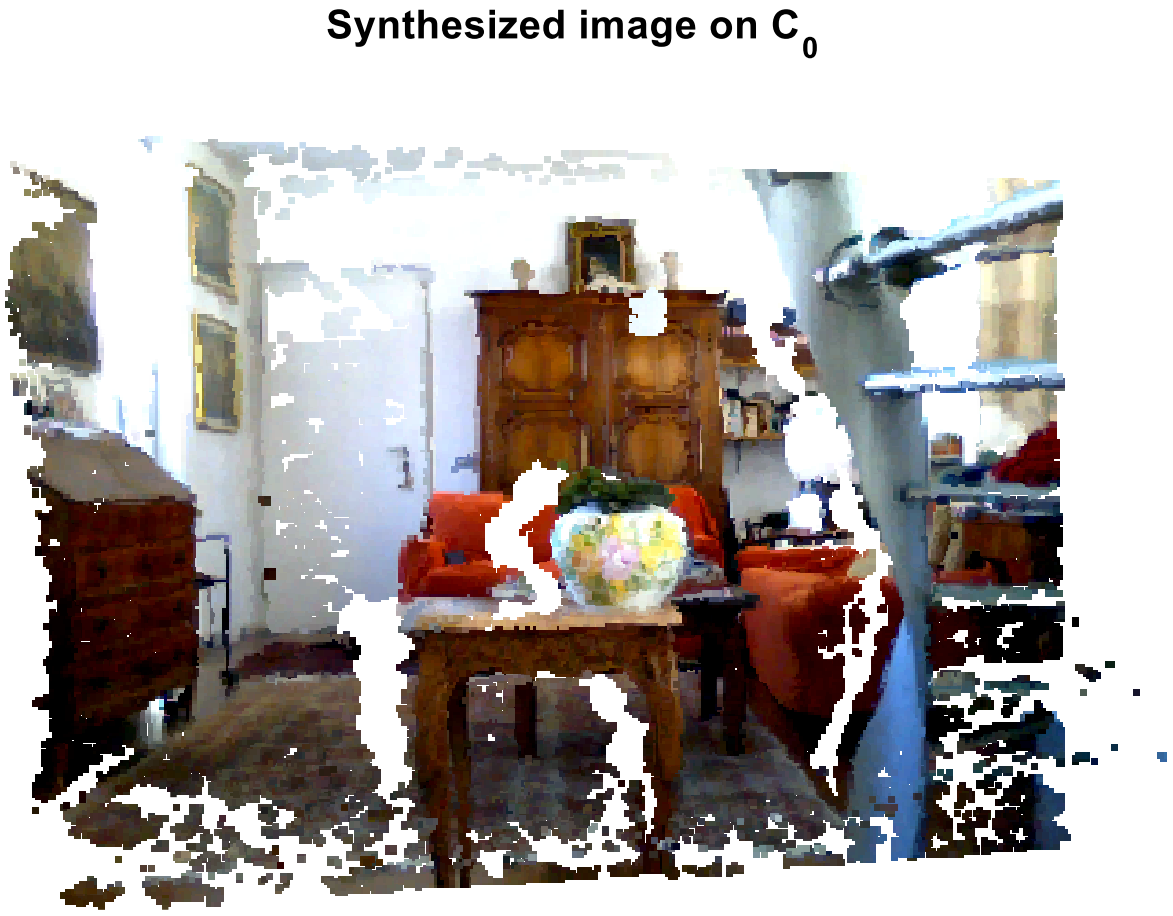


Figura 110 – Immagine sintetizzata Real-time, indoor, camere Mic, 640x480, ptsStep 2

6.5.4 – Test con risoluzione 320x240, camere Microsoft

SETTING DEI PARAMETRI GRAFICI PER OTTENERE L'IMMAGINE DI *Figura 111*		
ptsStep	sctrStep	scatterSize
1	1	5

Tabella 33 – Setting dei parametri grafici per ottenere l'immagine di *Figura 111*

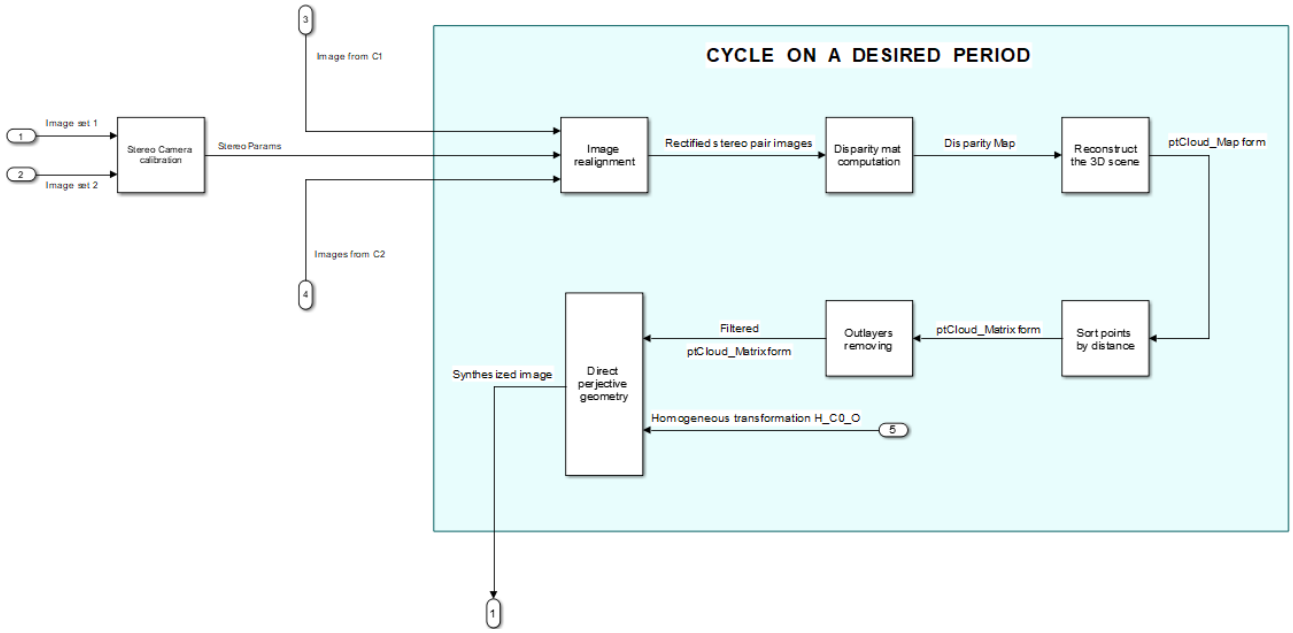
Figura 111 – Immagine sintetizzata VS acquisita, Real-time, indoor, camere Mic, 320x240

7 – CONCLUSIONI

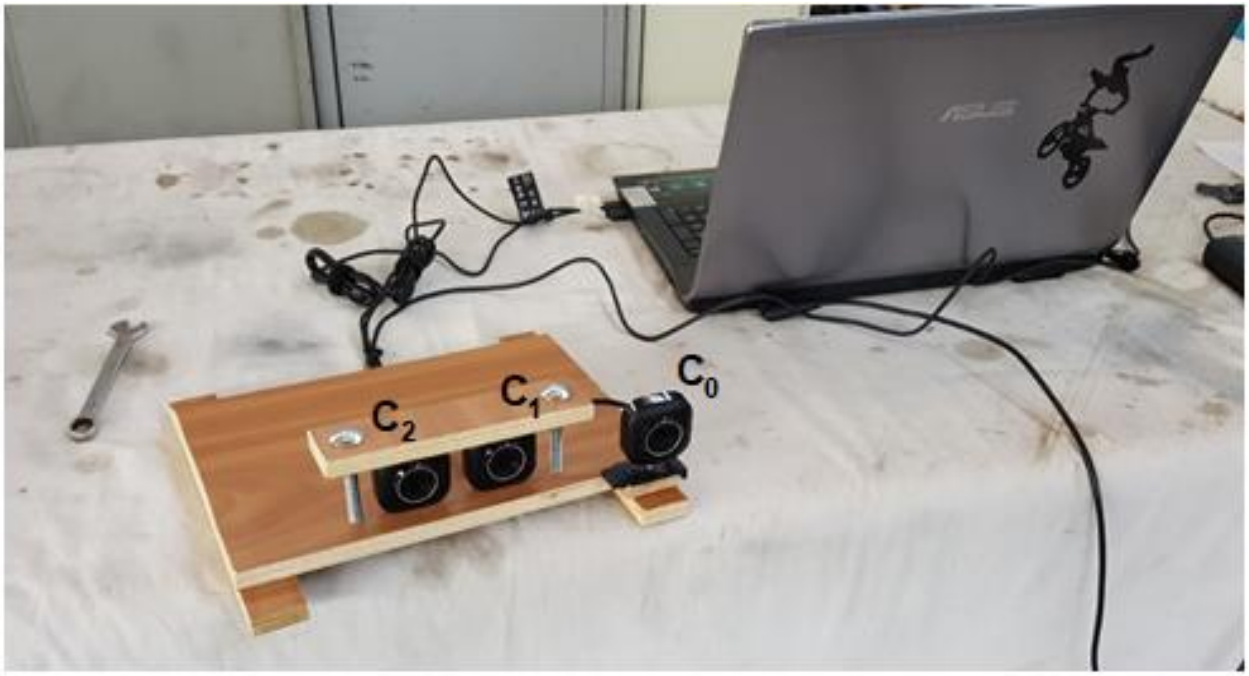
7.1 – Riassunto del lavoro svolto

E' esperienza comune ad ogni automobilista che la presenza dei montanti strutturali di tipo "A" possa talvolta compromettere seriamente la visibilità del conducente. Al fine di riprodurre al guidatore l'immagine che dovrebbe percepire in assenza del pilastro stesso, si sono valutati i primi aspetti tecnici scaturiti dall'eventuale installazione di un sistema a fotocamere-display.
Poiché il progetto del sistema non è ancora stato completamente terminato, buona parte del tempo rimanente è stato investito nel rendere il programma "il più portabile possibile".
Si tenga presente che molte delle immagini proposte sono state generate imponendo una qualità video piuttosto bassa: in questo modo si è consentito alla grafica di inseguire la più celere velocità del codice Matlab sviluppato.

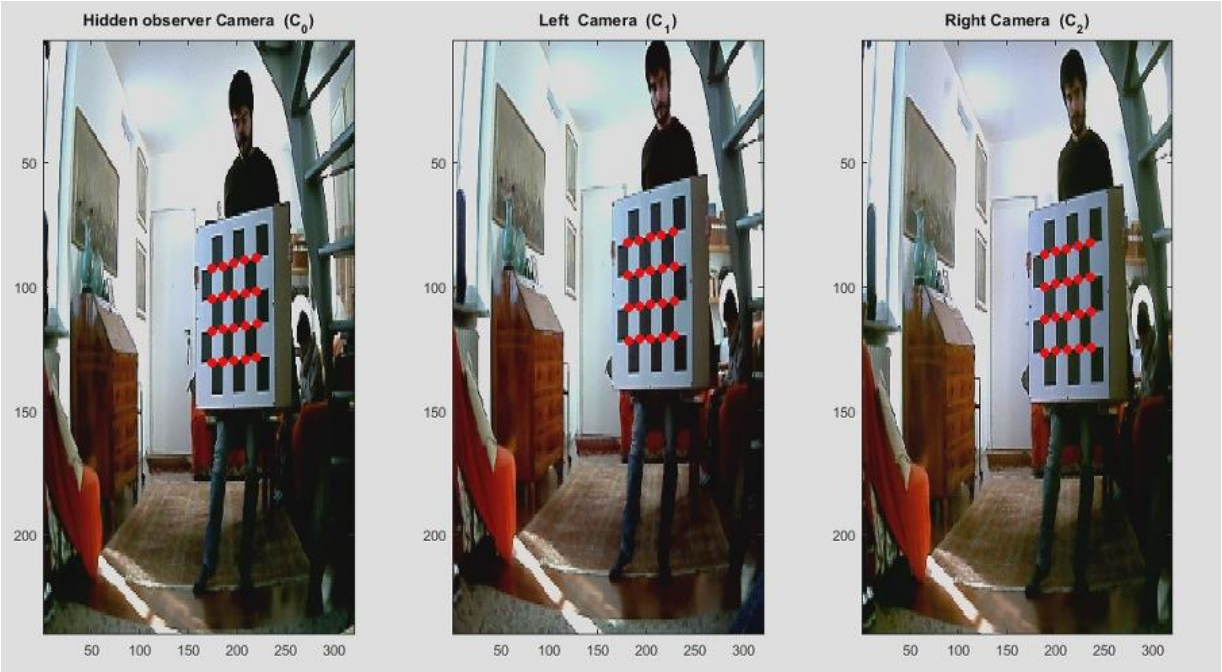
Tutti i programmi e test sperimentali illustrati nei primi capitoli, sono di fatto serviti alla comprensione profonda del codice descitto nel *Capitolo 6*. In detta sezione infatti, risiede la vera e propria essenza del programma di generazione immagini, il cui schema è illustrato nella seguente figura:



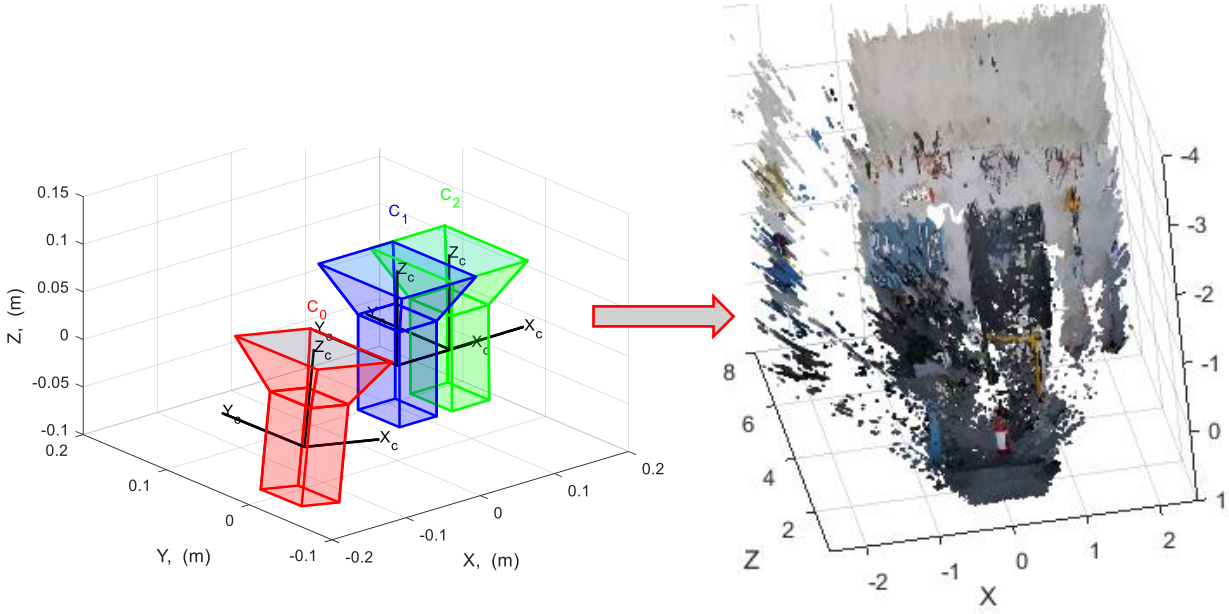
Come anticipato all'interno dell'*Abstract*, essendo il sistema attualmente ancora in fase di progetto, il tempo a disposizione è stato investito nel confronto tra l'immagine sintetizzata e l'immagine acquisita da una camera ("C_0") posta a fare le veci dell'osservatore nascosto dal montante A. Indipendentemente da ciò, si è reso necessario lo sviluppo di un setup orientato all'ottenimento della scena tridimensionale in cui si vuole installare il sistema. Per fare ciò, si è realizzata una stereo camera, costituita dal semplice affiancamento delle webcam ("C_1" & "C_2"):



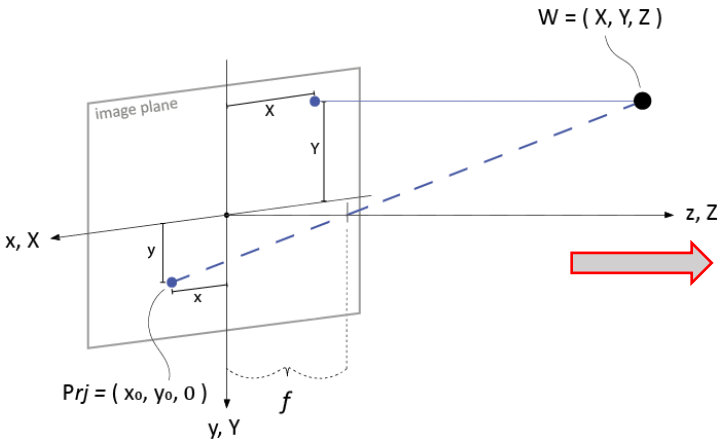
Per verificare la corretta sintesi dell'immagine invece, si è fatto affidamento al confronto tra l'immagine generata secondo il modello di "*pinhole camera*" e l'immagine realmente acquisita da C_0. Tali processi, hanno richiesto la preventiva conoscienza della locazione relativa delle tre camere ulitizzate. Per fare ciò, si è maturato uno script Matlab atto ad eseguire una calibrazione rapida delle camere:



Noto l'orientamento relativo delle camere "ausiliarie" al 3D (C_1 e C_2), si è ricostruita la scena tridimensionale dell'ambiente:



Ottenuta la scena tridimensionale, si è utilizzato il modello matematico della fotocamera abbinato ad un algoritmo di ordinamento (applicato alla "nuvola" di punti) per sintenizzare l'immagine che dovrebbe percepire l'osservatore C_0:



$$x = f\frac{X}{f - Z} \quad ; \quad y = f\frac{Y}{f - Z}$$

Il tutto con le tempistiche raccolte nella successiva tabella:

RACCOLTA TEMPI NECESSARI ALLA SINTESI DELL'IMMAGINE	
DATI | Webcam : Microsoft 320x240, Pixels processati: 76800, Computer: Asus N53S	
PROCESSO	**TEMPO, (s)**
Images loading	0.002
Images rectification	0.009
Disparity computation	0.100
Getting world points	0.009
Reducing points density	0.007
From map form to nx6	0.013
Sorting pts by distance	0.015
Filtering outlayers	0.001
Perspective transform	0.008
Time to pixels on	ND
TOTAL TIME NEEDED	**0.164 + ND**

Di seguito l'estrazione di un fotogramma esplicativo del confronto tra l'immagine realmente acquisita dall'osservatore nascosto (camera C_0) e quella "riproiettata" mediante le equazioni del modello matematico *pinhole camera*.

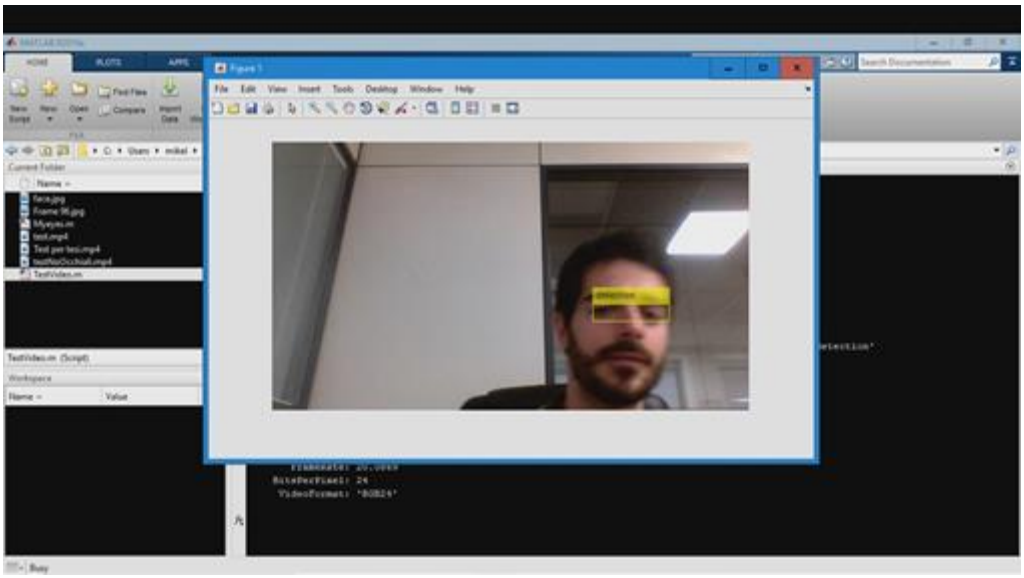
7.2 – Prosecuzione del progetto, limiti riscontrati e possibili migliorie

Nella presente trattazione non si è dato particolare peso al fatto che le immagini sintetizzate abbiano il corretto rapporto "*Ratio"* tra le scale degli assi orizzontali e verticali: è stata reputata una perdita di tempo accanirsi a fare coincidere i *Ratio* tra l'immagine sintetizzata e quella acquisita. Questo, per il semplice fatto che, in stadi di avanzamento futuri, dovrà essere il guidatore ad impostare tale valore prima di mettersi alla guida.

Si noti che l'ultimo blocco funzionale del diagramma a blocchi, possiede ad input l'oggetto "*ptCloud*" (descrittivo dell'ambiente circostante) e "*Homogeneous transformation H_C0_O*". Quest'ultimo, nella trattazione è descrittivo delle coordinate del centro della fotocamera C_0, posta a fare le veci dell'osservatore nascosto. In studi futuri, esso dovrà invece essere la descrizione della posizione ed orientamento degli occhi del guidatore, ipotizzati come una singola entità. Ulteriore parametro di input alla "*Direct projective geometry*" dovrà essere la "*lunghezza focale equivalente*" dell'occhio umano del guidatore.

Poiché la lunghezza focale equivalente di un occhio umano varia in ogni individuo e a seconda di "verso cosa" la mente è concentrata in quel momento, non si potrà a priori avere un sistema in grado di riprodurre la corretta immagine in ogni circostanza. Occorre quindi accontentarsi di impostare ad input (magari per mezzo di una semplice "manopola" sul cruscotto) una certa lunghezza focale, "tarata" in modo da funzionare correttamente negli specifici casi ritenuti dal guidatore maggiormente critici (ad esempio, durante l'ingresso in rotonda).

Il guidatore è proprio uno dei grossi problemi che impone il continuo aggiornamento dell'immagine (oltre che alla velocità relativa tra veicolo e resto del mondo) in quanto vi è la possibilità che il conducente "ruoti" i propri occhi oppure che li sposti a seguito di una variazione di posizione della sua testa. Per questa ragione, si rende necessario l'uso di una *eyes-detection* in Real-time. Come anticipato, essa darà luogo in ogni istante all'input *Homogeneous transformation H_C0_O* del blocco *Direct projective geometry*:



Si tenga presente che l'algoritmo di rilevamento degli occhi introdurrà ulteriori ritardi nel sistema. Si può tuttavia pensare di ridurre il problema mediante un sistema di misura di posizione che non sia di natura visiva: basato ad esempio sull'uso di appositi oggetti applicabili al viso e rilevabili elettronicamente da un altro dispositivo all'interno dell'abitacolo.

Un'altra possibile fonte di ritardi è lo scambio di dati tra il display/proiettore ed il pc generante il codice dell'immagine sintetizzata. Di meno importanza ma degno di nota, il fatto che ciò che percepisce il guidatore, è totalmente diverso da ciò che vede il passeggero. Pertanto, il passeggero potrebbe essere infastidito da un video totalmente incoerente con il resto dello scenario percepito attraverso il parabrezza.

La stima del 3D può essere migliorata mediante l'uso di sistemi di visione differenti. A tal proposito, è possibile pensare alla camera chiamata *Microsoft Kinect*. La prima versione era costituita da una camera RGB e una a radiazione infrarossa (aventi risoluzione di 480p ciascuna), caratterizzata da una frequenza di lettura pari a 30 fps. Per il calcolo della profondità, Kinect si avvaleva di uno scanner 3D a luce strutturata, in combinazione con infrarossi. Occorre però tenere in considerazione che Kinect e dispositivi sifatti non lavorano bene in prossimità di ambienti con sole, pertanto la sua installazione all'esterno di un auto non è necessariamente una buona idea. Altra possibile miglioria consiste nell'aggiunta di ulteriori fotocamere: in questo modo si otterranno mappe delle disparità più complete, conseguentemente immagini sintetizzate più ricche di punti ed affidabili.

É possibile pensare di compensare il ritardo di generazione dell'immagine graficando ad ogni ciclo temporale un'immagine figlia dell'immagine acquisita e della velocità di cambiamento dei dati. In altre parole, se ci fossero corpi in movimento all'interno dell'immagine, è possibile pensare di rilevarli matematicamente e prevederne la posizione al tempo futuro in base alla loro velocità. Di contro, si perderebbe in affidabilità e probabilmente il dispendio temporale apportato dall'aggiunta di tale calcolo non necessariamente avrebbe effetti benefici sul tempo totale.

Si consideri, infine, che il *Sistema di identificazione* proposto nella presente trattazione non è necessariamente circosritto all'ambito della circolazione stradale. Esso potrebbe infatti portare ad una maggiore sicurezza o a miglioramenti negli ambiti industriale, aeronautico, di sorveglianza, sanitario, ecc.

Michele Pecchio Ghiringhelli Rota

8 – BIBLIOGRAFIA

- Al Volante ---------- : **La Jaguar sperimenta i "montanti trasparenti"**, Jaguar e Land Rover, Al Volante
- HD motori ---------- : **Toyota brevetta i montanti "trasparenti"**, Toyota
- Yu-Lin Chang, Yi-Min Tsai, Liang-Gee Chen ---: **A Real-Time Augmented View Synthesis System for transparent car pillars**, National Taiwan University, Taipei, Taiwan
- Key Oishi, Shohei Mori, Hideo Saito ---------- : **An Instant See-Through Vision System Using a Wide Field-of-View Camera and a 3D-Lidar**, 2017 IEEE International Symposium on Mixed and Augmented Reality Adjunct Proceedings
- Beresnev, Tumasov, Zeziulin, Porubov, Orlov: **Development of a detection road users system in vehicle A-pillar blind spots**, Nizhny Novgorod Stare Technical University n. a. R. E. Alekseev, Russian Federation
- Prof. P. K. Biswas ---------- : **Digital Image Processing, Camera Model and Imaging Geometry**, Youtube, Lecture 7, Indian Institute of Technology Kharagpur & MHRD, Govt. of India
- Peter Corke ---------- : **Image geometry and planar homography**, Youtube, Introduction to Machine Vision, ENB339
- Dr. Mubarak Shah ---------- : **Intorduction to Computer Vision**, Youtube, UCF Computer Vision Lectures 2012, University of central Florida, Florida
- Rich Radke ---------- : **Stereo correspondence**, Youtube, CVFX Lecture 15, Rensselaer Polytechnic Institute, Troy

- Angela Sodemann ---------- : **Homogeneous Transformation Matrix Example and Coordinate Transformation**, Youtube, Robotics, Lecture Video
- Sanja Fidler ---------- : **Depth from Stereo CSC420**, Intro to image understanding, Computer Science, University of Toronto
- Fusiello, Trucco, Verri, e successivi ---------- : **Image rectification**, Computer Science, Computer Stereo Vision, Wikipedia
- Chris McCormick ---------- : **Stereo Vision Tutorial**, Part I, Nearist
- University of Tsukuba ---------- : **Tsukuba**, Stereo pair Image, University of Tsukuba, Ibaraki, Japan
- G. Bradski, A. Kaehler, ---------- : **Depth Estimation From Stereo Video**, MathWorks and Learning OpenCV, Matlab Documentation
- Microsoft ---------- : **Microsoft Kinect**, Wikipedia, Electronic Entertainment Expo
- Zhang, Z.; Heikkila, J., Silven ---------- : **stereoParameters**, MathWorks, Matlab Documentation
- Z, Z; H, J, S; Bouguet, J.Y. Bradski, G.,Kaehler : **What is Camera Calibration?**, MathWorks, Matlab Documentation

9 – ALLEGATI

9.1 – Listati Matlab

9.1.1 – Rebld_F_Free_Cams.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%     "F" REPROJECTION ON WALL AND DISPLAY       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        Michele Pecchio Ghiringhelli Rota       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            Last update: 18/12/2018             %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% INITIALIZATION ==================================================================================

clear;
clc;
close all
addpath(genpath('FunctionFolder'))       % Now all functions are avaiable for the call...



%% DATA ============================================================================================

% Number of Cameras
N = 3;                                   % Not settable now

% Camera 0 Data:
dOC0  = [.2 .8 -.6];                     % Coordinates of the camera C0 whit respect to the
                                         % {X,Y,Z;O} System
alfa0 = pi/20;                           % Rotation angle around the original Z axis, (rad)
beta0 = pi/18;                           % Rotation angle around the original Y axis, (rad)
gama0 = pi/2;                            % Rotation angle around the original X axis, (rad)
f(1)  =  .62;                            % Focal length, (m). Remember that human focal lenght is
                                         % about 40~60 mm

% Camera 1 Data:
dOC1  = [-.31 1.02 .155];                % Coordinates of the camera C1 whit respect to the
                                         % {X,Y,Z;O} System
alfa1 = pi/10;                           % Rotation angle around the original Z axis, (rad)
beta1 = -pi/15;                          % Rotation angle around the original Y axis, (rad)
gama1 = pi/2;                            % Rotation angle around the original X axis, (rad)
f(2)  = .65;                             % Focal length, (m)

% Camera 2 Data:
dOC2  = [1.31 1.02 .355];                % Coordinates of the camera C2 whit respect to the
                                         % {X,Y,Z;O} System
alfa2 = pi/7;                            % Rotation angle around the original third  axis of C1
                                         % system, (rad)
beta2 = -pi/8;                           % Rotation angle around the original second axis of C1
                                         % system, (rad)
gama2 = -pi/15;                          % Rotation angle around the first _________ axis of C1
                                         % system, (rad)
f(3)  = .61;                             % Focal length, (m)

% Common Data:
Matrix_F = generate_F_letter(1,1);       % Crop all points of F into a Matrix. FirstInput = size
CamSize  = .2;                           % Size of the Cameras on the plot
nPoints  = size(Matrix_F,1);             % Number of 3D World points
```

```
%% TRANSFORMATION MATRIX  ========================================================================

% Translation Matrix from the absolute Origin to the Camera origin, with respect {X,Y,Z;O} sys. 4X4
% Translation Matrix
Tr0_MATRIX = GenTrMatrix(dOC0);
Tr1_MATRIX = GenTrMatrix(dOC1);

% Positive rotation Matrix around the original x axis: CCW direction with x verse which poins you
[Rx0_Matrix, Rx0_MATRIX] = GenRxMatrix(gama0);
[Rx1_Matrix, Rx1_MATRIX] = GenRxMatrix(gama1);
[Rx2_Matrix, Rx2_MATRIX] = GenRxMatrix(gama2);

% Positive rotation Matrix around the original y axis: CCW direction with y verse which poins you
[Ry0_Matrix, Ry0_MATRIX] = GenRyMatrix(beta0);
[Ry1_Matrix, Ry1_MATRIX] = GenRyMatrix(beta1);
[Ry2_Matrix, Ry2_MATRIX] = GenRyMatrix(beta2);

% Positive rotation Matrix around the original z axis: CCW direction with z verse which poins you
[Rz0_Matrix, Rz0_MATRIX]  = GenRzMatrix(alfa0);
[Rz1_Matrix, Rz1_MATRIX]  = GenRzMatrix(alfa1);
[Rz2_Matrix, Rz2_MATRIX]  = GenRzMatrix(alfa2);



%% PROJECTIVE GEOMETRY  ========================================================================

% Remember the following laws: R_Matrix^-1 = R_Matrix^T. On the contrary,
% Tr_Matrix^-1 ~= Tr_Matrix^T

% Preallocating
Cam_coords = zeros(nPoints,4,N);
Img_coords = zeros(nPoints,3,N);
PrjBaseC   = zeros(nPoints,4,N);
PrjBaseO   = zeros(nPoints,4,N);
figure(1); title('3D Setup'); grid on; hold on;
xlabel('X,  (m)'); ylabel('Y,  (m)'); zlabel('Z,  (m)');

% Understanding Cameras Location and Orientation.
FirstRC0O  = Rx0_MATRIX;                                          % First  rotation of camera C0 from
                                                                  % the configuration on the origin
SecondRC0O = Ry0_MATRIX;                                          % Second rotation of camera C0 from
                                                                  % the First  rotation
ThirdRC0O  = Rz0_MATRIX;                                          % Third  rotation of camera C0 from
                                                                  % the Second rotation
HomogC0O   = (ThirdRC0O * SecondRC0O * FirstRC0O) / Tr0_MATRIX; % Homogeneous transformation from
                                                                  % the Camera C0 system to the O
                                                                  % system
HomogOC0   = inv(HomogC0O);                                       % Homogeneous transformation from
                                                                  % the system O to the Camera C0
                                                                  % system
Cam0       = plotCamera('Location',dOC0,'Orientation',HomogC0O(1:3,1:3),'Opacity',0.15,...
                        'AxesVisible',true,'Color','r','Size',CamSize);

FirstRC1O  = Rx1_MATRIX;                                          % It goes the same for this
                                                                  % Camera...
SecondRC1O = Ry1_MATRIX;
ThirdRC1O  = Rz1_MATRIX;
HomogC1O   = (ThirdRC1O * SecondRC1O * FirstRC1O) / Tr1_MATRIX; % Equivalent to: SecondRC1O *
                                                                  % FirstRC1O * inv(Tr1_MATRIX)
HomogOC1   = inv(HomogC1O);
Cam1       = plotCamera('Location',dOC1,'Orientation',HomogC1O(1:3,1:3),'Opacity',0.15,...
                        'AxesVisible',true,'Color','g','Size',CamSize);

FirstRC2C1  = Rx2_MATRIX';                                        % First  rotation of Camera C2 from
                                                                  % the configuration   of C1 camera
SecondRC2C1 = Ry2_MATRIX';                                        % Second rotation of Camera C2 from
                                                                  % the first rotation  of C1 camera
ThirdRC2C1  = Rz2_MATRIX';                                        % Third  rotation of Camera C2 from
                                                                  % the second rotation of C1 camera
```

```
HomogC2C1    = (FirstRC2C1 * SecondRC2C1 * ThirdRC2C1) / ([eye(4,3) HomogC1O*[dOC2 1]']);
HomogC1C2    = inv(HomogC2C1);

HomogC2O     = (HomogC2C1*HomogC1O);                                % Homogeneous transformation from
                                                                    % the Camera C2 system to the O
                                                                    % system
HomogOC2     = inv(HomogC2O);                                       % Homogeneous transformation from
                                                                    % the O system to the Camera C2
                                                                    % system
Cam2         = plotCamera('Location',dOC2,'Orientation',HomogC2O(1:3,1:3),'Opacity',0.15,...
                          'AxesVisible',true,'Color','b','Size',CamSize);

% Data managment for the following plot
Struct(1).HomogOC = HomogOC0;    % "(1)" --> Homog. transf. of the "C0" Camera. From the origin to C0
Struct(2).HomogOC = HomogOC1;    % "(2)" --> Homog. transf. of the "C1" Camera. From the origin to C1
Struct(3).HomogOC = HomogOC2;    % "(3)" --> Homog. transf. of the "C2" Camera. From the origin to C2

Struct(1).HomogCO = HomogC0O;    % "(1)" --> Homog. transf. of the "C0" Camera. From C0 to the origin
Struct(2).HomogCO = HomogC1O;    % "(2)" --> Homog. transf. of the "C1" Camera. From C1 to the origin
Struct(3).HomogCO = HomogC2O;    % "(3)" --> Homog. transf. of the "C2" Camera. From C2 to the origin

Struct(1).Color   = [.7 0 0];    % Color plot associated with C0 Camera
Struct(2).Color   = [0 .7 0];    % Color plot associated with C1 Camera
Struct(3).Color   = [0 0 .7];    % Color plot associated with C2 Camera

% Find the Projection of all "F" points on each Cameras Image planes. Follows a 3D plot
for jj = 1 : N

    for ii = 1 : nPoints

        % Coordinates of each "F points" with respect to Camera system
        Cam_coords(ii,:,jj) = Struct(jj).HomogCO * [Matrix_F(ii,:) 1]';

        % Coordinates of projected "F points" with respect to image planes
        Img_coords(ii,:,jj) = [f(jj)*Cam_coords(ii,1,jj)/(f(jj)-Cam_coords(ii,3,jj))...
                               f(jj)*Cam_coords(ii,2,jj)/(f(jj)-Cam_coords(ii,3,jj))...
                               f(jj)*Cam_coords(ii,3,jj)/(f(jj)-Cam_coords(ii,3,jj))];

        PrjBaseC(ii,:,jj) = [Img_coords(ii,1:2,jj) 0 1];
        PrjBaseO(ii,:,jj) = Struct(jj).HomogOC  *  PrjBaseC(ii,:,jj)';

        if jj > .1   % Set less than 1 to highlithing the first Camera points. Then, it will be a
future check.

scatter3(PrjBaseO(ii,1,jj),PrjBaseO(ii,2,jj),PrjBaseO(ii,3,jj),6,Struct(jj).Color,'o','Linewidth',1)
        end

    end

end

% Link coordinates of the projected "F points" for each Image plane
for jj = 1 : N  % Skip linking the first Camera lines.

    plot3([PrjBaseO(1,1,jj) PrjBaseO(2,1,jj)],[PrjBaseO(1,2,jj) PrjBaseO(2,2,jj)],[PrjBaseO(1,3,jj)
PrjBaseO(2,3,jj)],...
          'color',Struct(jj).Color,'Linewidth',1.5)
    plot3([PrjBaseO(2,1,jj) PrjBaseO(3,1,jj)],[PrjBaseO(2,2,jj) PrjBaseO(3,2,jj)],[PrjBaseO(2,3,jj)
PrjBaseO(3,3,jj)],...
          'color',Struct(jj).Color,'Linewidth',1.5)
    plot3([PrjBaseO(3,1,jj) PrjBaseO(4,1,jj)],[PrjBaseO(3,2,jj) PrjBaseO(4,2,jj)],[PrjBaseO(3,3,jj)
PrjBaseO(4,3,jj)],...
          'color',Struct(jj).Color,'Linewidth',1.5)
    plot3([PrjBaseO(2,1,jj) PrjBaseO(5,1,jj)],[PrjBaseO(2,2,jj) PrjBaseO(5,2,jj)],[PrjBaseO(2,3,jj)
PrjBaseO(5,3,jj)],...
          'color',Struct(jj).Color,'Linewidth',1.5)

end
```

```
%% INVERSE PERSPECTIVE GEOMETRY. NEED AT LEAST TWO CAMERAS. NOW WE FIND THE 3D WORLD POINTS OF "F"
%  FROM C1 & C2 POINTS ============================================================================

% Solve with Cramer method. We're now obtainig the World points coordinates with respect to C1
% camera by using also C2 datas

alfa = alfa2;                   % Orientation between Camera 1 and Camera 2. From system 1 to system
                                % 2, along z Camera axis
beta = beta2;                   % Orientation between Camera 1 and Camera 2. From system 1 to system
                                % 2, along y Camera axis
gama = gama2;                   % Orientation between Camera 1 and Camera 2. From system 1 to system
                                % 2, along x Camera axis


f1 = f(2);                      % Just to semplify the notation
f2 = f(3);                      % Just to semplify the notation

C2_C1  = (HomogC1O*[dOC2 1]')'; % Vector which crops all C2 components with respect to the C1 Camera
                                % sys


XC2_C1 = C2_C1(:,1);            % First  component
YC2_C1 = C2_C1(:,2);            % Second component
ZC2_C1 = C2_C1(:,3);            % Third  component

k1 = PrjBaseC(:,1,2)/f1;        % Just to semplify the notation
k2 = PrjBaseC(:,2,2)/f1;        % It goes the same...
k3 = PrjBaseC(:,1,3)/f2;
k4 = XC2_C1;
k5 = ZC2_C1;
k6 = YC2_C1;

s_a = sin(alfa);    c_a = cos(alfa);
s_b = sin(beta);    c_b = cos(beta);
s_g = sin(gama);    c_g = cos(gama);

j1 = -c_a*c_b;
j2 = -s_a*s_g-c_a*s_b*c_g;
j3 = s_a*c_g-c_a*s_b*s_g;
j4 = -s_a*c_b;
j5 = c_a*s_g-s_a*s_b*c_g;
j6 = -(s_a*s_b*s_g+c_a*c_g);
j7 = s_b;
j8 = -c_b*c_g;
j9 = -c_b*s_g;

%{
% Numerical solution

%   A = [1    0   k1    0    0    0
%        0    1   k2    0    0    0
%        0    0    0    1   k3    0
%        1    0    0   j1   j2   j3
%        0    1    0   j4   j5   j6
%        0    0    1   j7   j8   j9];

% b = [k1*f1  k2*f1  k3*f2  k4  k6  k5]';

A = zeros(6,6,nPoints);
% First (deep) row
A(1,1,:) = 1;
A(1,2,:) = 0;
A(1,3,:) = k1;
A(1,4,:) = 0;
A(1,5,:) = 0;
A(1,6,:) = 0;

% Second (deep) row
A(2,1,:) = 0;
A(2,2,:) = 1;
A(2,3,:) = k2;
A(2,4,:) = 0;
A(2,5,:) = 0;
A(2,6,:) = 0;
```

```
% Third (deep) row
A(3,1,:) = 0;
A(3,2,:) = 0;
A(3,3,:) = 0;
A(3,4,:) = 1;
A(3,5,:) = k3;
A(3,6,:) = 0;

% Fourth (deep) row
A(4,1,:) = 1;
A(4,2,:) = 0;
A(4,3,:) = 0;
A(4,4,:) = j1;
A(4,5,:) = j2;
A(4,6,:) = j3;

% Fifth (deep) row
A(5,1,:) = 0;
A(5,2,:) = 1;
A(5,3,:) = 0;
A(5,4,:) = j4;
A(5,5,:) = j5;
A(5,6,:) = j6;

% Sixth (deep) row
A(6,1,:) = 0;
A(6,2,:) = 0;
A(6,3,:) = 1;
A(6,4,:) = j7;
A(6,5,:) = j8;
A(6,6,:) = j9;

% Known therms
b(1,1,:) = PrjBaseC(:,1,2);
b(2,1,:) = PrjBaseC(:,2,2);
b(3,1,:) = PrjBaseC(:,1,3);
b(4,1,:) = XC2_C1;
b(5,1,:) = YC2_C1;
b(6,1,:) = ZC2_C1;

x_Vector = zeros(6,1,nPoints);
for ii = 1 : nPoints

    x_Vector(:,1,ii) = A(:,:,ii) \ b(:,1,ii);

    tt = ii/nPoints*100;
    fprintf('Processing element %d/%d... Percntge = %.2f %%\n ', ii,nPoints,tt);

end

% Coordinates of each point of matrix "Matrix_F" with respect to the camera C1 system
XBaseC1 = reshape(x_Vector(1,1,:),nPoints,1);
YBaseC1 = reshape(x_Vector(2,1,:),nPoints,1);
ZBaseC1 = reshape(x_Vector(3,1,:),nPoints,1);

PBaseC1 = [XBaseC1 YBaseC1 ZBaseC1 ones(nPoints,1)];
%}

%{%
% Cramer analytical solution
Detrmnt = (j3*j5-j2*j6) + k3*(j1*j6-j3*j4) + k2*(j3*j8-j2*j9) + k2.*k3*(j1*j9-j3*j7) + ...
           k1*(j5*j9-j6*j8) + k1.*k3*(j6*j7-j4*j9);

XBaseC1 = (-k1*f1*(j2*j6-j3*j5) + k1.*k3*f1*(j1*j6-j3*j4) -k1.*k3*f2*j1*(j5*j9-j6*j8) + ...
            k1.*k3*f2*j2*(j4*j9-j6*j7) -k1.*k3*f2*j3*(j4*j8-j5*j7) + k1*k4*(j5*j9-j6*j8) + ...
           -k1*j2*(k6*j9-k5*j6) + k1*j3*(k6*j8-k5*j5) -k1.*k3*k4*(j4*j9-j6*j7) +...
            k1.*k3*j1*(k6*j9-k5*j6) - k1.*k3*j3*(k6*j7-k5*j4)) ./ Detrmnt;

YBaseC1 = (-k2*f1*(j2*j6-j3*j5) + k2.*k3*f1*(j1*j6-j3*j4) - k2.*k3*f2*j1*(j5*j9-j6*j8) + ...
            k2.*k3*f2*j2*(j4*j9-j6*j7) -k2.*k3*f2*j3*(j4*j8-j5*j7) + k2*k4*(j5*j9-j6*j8)+ ...
           -k2*j2*(k6*j9-k5*j6) + k2*j3*(k6*j8-k5*j5) -k2.*k3*k4*(j4*j9-j6*j7) + ...
            k2.*k3*j1*(k6*j9-k5*j6) - k2.*k3*j3*(k6*j7-k5*j4)) ./ Detrmnt;
```

```
ZBaseC1 = (+k3*f2*j1*(j5*j9-j6*j8) - k3*f2*j2*(j4*j9-j6*j7) + k3*f2*j3*(j4*j8-j5*j7) + ...
           -k4*(j5*j9-j6*j8) +j2*(k6*j9-k5*j6) - j3*(k6*j8-k5*j5) + k3*k4*(j4*j9-j6*j7) + ...
           -k3*j1*(k6*j9-k5*j6) +k3*j3*(k6*j7-k5*j4) -k2*f1*(j2*j9-j3*j8) + ...
            k2.*k3*f1*(j1*j9-j3*j7) + k1*f1*(j5*j9-j6*j8) - k1.*k3*f1*(j4*j9-j6*j7)) ./ Detrmnt;

% Coordinates of each point of matrix "Matrix_F" with respect to the camera C1 system
PBaseC1 = [XBaseC1 YBaseC1 ZBaseC1 ones(nPoints,1)];
%}

% Coordinates of each point of matrix "Matrix_F" with respect to the base O system
PBaseO = HomogOC1*PBaseC1';

% Reproject the founded points to have a graphical feedback
scatter3(PBaseO(1,:),PBaseO(2,:),PBaseO(3,:),4,'c','Linewidth',1.8)



%% GETTING THE IMAGE PLANE 0 IMAGE. RECONSTRUCTION ALGORITHM ======================================

% Coordinates of each "F points" with respect to C0 Camera system
clear Cam_coords;
Cam_coords = HomogC0O * PBaseO;
Cam_coords = Cam_coords';

% Coordinates of projected "F points" with respect to image plane 0
Img_coords = [f(1)*Cam_coords(:,1)./(f(1)-Cam_coords(:,3))...
              f(1)*Cam_coords(:,2)./(f(1)-Cam_coords(:,3))...
              f(1)*Cam_coords(:,3)./(f(1)-Cam_coords(:,3))];

figure(2); grid on; hold on
title(['Image plane ' num2str(0) '. Focal length = ' num2str(f(1)) ' m']); xlabel('x,  (m)');
ylabel('y,  (m)')

% Reproject the founded points to have a graphical feedback
for ii = 1 : nPoints
    scatter(Img_coords(ii,1),Img_coords(ii,2),20,Struct(1).Color);
end

% Draw each line
plot([Img_coords(1,1) Img_coords(2,1)],[Img_coords(1,2)
Img_coords(2,2)],'color',Struct(1).Color,'Linewidth',1.5)
plot([Img_coords(2,1) Img_coords(3,1)],[Img_coords(2,2)
Img_coords(3,2)],'color',Struct(1).Color,'Linewidth',1.5)
plot([Img_coords(3,1) Img_coords(4,1)],[Img_coords(3,2)
Img_coords(4,2)],'color',Struct(1).Color,'Linewidth',1.5)
plot([Img_coords(2,1) Img_coords(5,1)],[Img_coords(2,2)
Img_coords(5,2)],'color',Struct(1).Color,'Linewidth',1.5)

axis equal



%% _END PROGRAM_
```

9.1.2 – generate_F_letter.m

```
function [ MatrixP ] = generate_F_letter( size , figure)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

    wd = 1.96; % Wall depth, (m)

    P1 = [.4   wd .43];
    P2 = [.4   wd .55];
    P3 = [.4   wd .63];
    P4 = [.545 wd .63];
    P5 = [.505 wd .55];

    MatrixP = [P1; P2; P3; P4; P5];
    MatrixP = size*MatrixP;

    if figure == 1
        plot3([P1(1) P3(1)],[P1(2) P3(2)],[P1(3) P3(3)],'k','Linewidth',2); grid on; hold on;
        plot3([P2(1) P5(1)],[P2(2) P5(2)],[P2(3) P5(3)],'k','Linewidth',2)
        plot3([P3(1) P4(1)],[P3(2) P4(2)],[P3(3) P4(3)],'k','Linewidth',2)
    end

end
```

9.1.3 – GenTrMatrix.m

```
function [ Tr_Matrix ] = GenTrMatrix( v )
%GenTrMatrix Generate a translation Matrix of v vector
Tr_Matrix = [1         0         0    v(1);    % Translation Matrix from the (X,Y,Z) Origin to
                                               % the Camera origin
             0         1         0    v(2);
             0         0         1    v(3);
             0         0         0      1];
end
```

9.1.4 – GenRxMatrix.m

```
function [ Rx_Matrix, Rx_MATRIX ] = GenRxMatrix( alfa )
%GenRxMatrix Generate a Rotation Matrix of alfa angle around x axis

zeri = [0 0 0];                                 % To make the homogeneous matrix

Rx_Matrix = [1         0         0       ;     % Positive rotation Matrix around x axis: CCW
                                               % direction with x verse which points you
             0     cos(alfa)    -sin(alfa)  ;
             0     sin(alfa)     cos(alfa)  ];

Rx_MATRIX = [Rx_Matrix  zeri';                  % 4X4 Rotation Matrix
                zeri     1  ];
end
```

9.1.5 – GenRyMatrix.m

```
function [ Ry_Matrix, Ry_MATRIX ] = GenRyMatrix( beta )
%GenRyMatrix Generate a Rotation Matrix of alfa angle around y axis

zeri = [0 0 0];                                     % To make the homogeneous matrix

Ry_Matrix = [cos(beta)    0      sin(beta) ;       % Positive rotation Matrix around y axis: CCW
                                                   % direction with y verse which points you
                0         1          0     ;
            -sin(beta)    0      cos(beta) ];

Ry_MATRIX = [Ry_Matrix  zeri';                      % 4X4 Rotation Matrix
                zeri     1  ];
end
```

9.1.6 – GenRzMatrix.m

```
function [ Rz_Matrix, Rz_MATRIX ] = GenRzMatrix( gama )
%GenRzMatrix Generate a Rotation Matrix of alfa angle around z axis

zeri = [0 0 0];                                    % To make the homogeneous matrix

Rz_Matrix = [cos(gama)    -sin(gama)    0 ;       % Positive rotation Matrix around z axis: CCW
direction with z verse which points you
             sin(gama)     cos(gama)    0 ;
                0             0        1 ];

Rz_MATRIX = [Rz_Matrix  zeri';                     % 4X4 Rotation Matrix
                zeri     1  ];

end
```

9.1.7 – Rebld_Img_Free_Cams.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%         IMAGE REPROJECTION ON A DISPLAY        %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        Michele Pecchio Ghiringhelli Rota       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%             Last update: 27/01/2019            %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% !! Remember that the following code has to be completed if C1 and C2 Cameras are not aligned !!  %

%% INITIALIZATION ==================================================================================

clear
clc
close all
addpath(genpath('FunctionFolder'));      % Now all functions are avaiable for the call...



%% DATA ============================================================================================

% Image loading
IL = imread('left.jpg');
IR = imread('right.jpg');

% Image reduction
%{
nR_Red = 240;
nC_Red = 320;
IL = uint8(IL(1:nR_Red,1:nC_Red,:));
IR = uint8(IR(1:nR_Red,1:nC_Red,:));
%}

% Common data
PhotoWidth = .16;                        % Width of the photos, (m)
tHold      = 2;                          % Sigma/mean coefficient. It can change over the following
code
CamSize    = .05;                        % Size of the Cameras on the plot
BlockSize  = 15;                         % Size of the matching windows

% Camera 1 Data:
dOC1   = [0 0 0];                        % Coordinates of the camera C1 whit respect to the
                                         % {X,Y,Z;O} System
alfa1 = 0;                               % Rotation angle around the original Z axis, (rad)
beta1 = 0;                               % Rotation angle around the original Y axis, (rad)
gama1 = 0;                               % Rotation angle around the original X axis, (rad)
f1     = 35/1000;                        % Focal length of C1 Camera, (m)

% Camera 2 Data:
alfa2 = 0;                               % Rotation angle around the original third  axis of C1
                                         % system, (rad)
beta2 = 0;                               % Rotation angle around the original second axis of C1
                                         % system, (rad)
gama2 = 0;                               % Rotation angle around the first _________ axis of C1
                                         % system, (rad)
f2     = 35/1000;                        % Focal length of C2 Camera, (m)
TrC2_C1 = [.2 0 0];                      % Coordinates of C2 Camera with respect to C1 Camera system

% Camera 0 Data:
alfa0 = 0;                               % Rotation angle around the original Z axis, (rad)
beta0 = -pi/12;                          % Rotation angle around the original Y axis, (rad)
gama0 = 0;                               % Rotation angle around the original X axis, (rad)
f0     =  35/1000;                       % Focal length of the hidden observer, (m). Remember that
                                         % human focal lenght is about 40~60 mm
TrC0_C1 = [-.2 0 -.2];                   % Coordinates of C0 Camera with respect to C1 Camera system
```

```
%% PRELIMINARY CALCULATIONS ================================================================================

% Manage of the left image
[nRows, nColu] = size(IL(:,:,1));
conv = 1; % PhotoWidth/nColu;

% Homogeneous Matrices
TrOC1_MATRIX = GenTrMatrix(dOC1);

[Rx0_Matrix, Rx0_MATRIX] = GenRxMatrix(gama0);
[Rx1_Matrix, Rx1_MATRIX] = GenRxMatrix(gama1);
[Rx2_Matrix, Rx2_MATRIX] = GenRxMatrix(gama2);

[Ry0_Matrix, Ry0_MATRIX] = GenRyMatrix(beta0);
[Ry1_Matrix, Ry1_MATRIX] = GenRyMatrix(beta1);
[Ry2_Matrix, Ry2_MATRIX] = GenRyMatrix(beta2);

[Rz0_Matrix, Rz0_MATRIX] = GenRzMatrix(alfa0);
[Rz1_Matrix, Rz1_MATRIX] = GenRzMatrix(alfa1);
[Rz2_Matrix, Rz2_MATRIX] = GenRzMatrix(alfa2);

% Homogeneous Transformation, betwheen C1 camera and O origin
alpha = 0;
RxMatrix = rotx(alpha);
RxMATRIX = [RxMatrix [0 0 0]'; [0 0 0 1]];
HomogOC1 = RxMATRIX;
HomogC1O = inv(HomogOC1);

% Homogeneous Transformation, betwheen C1 camera and C0 camera
FirstRC1C0  = Rx0_MATRIX;
SecondRC1C0 = Ry0_MATRIX;
ThirdRC1C0  = Rz0_MATRIX;
R_C1_C0     = ThirdRC1C0 * SecondRC1C0 * FirstRC1C0;
T_C0_C1     = GenTrMatrix(TrC0_C1);                         % Units: (m)
HomogC1C0   = R_C1_C0/T_C0_C1;                              % Equivalent to: R_C1_C0*inv(T_C0_C1),
which is R_C1_C0*T_C1_C0
HomogC0C1   = inv(HomogC1C0);                               % Units: (m) and (rad)
dOC0        = HomogOC1\[TrC0_C1 1]';                        % Equivalent to: HomogC1O*
                                                            % [-paramsC1_C0.pa...

% Homogeneous Transformation, betwheen C1 camera and C2 camera
FirstRC1C2  = Rx2_MATRIX;                                   % It goes the same for this Camera...
SecondRC1C2 = Ry2_MATRIX;
ThirdRC1C2  = Rz2_MATRIX;
R_C1_C2     = ThirdRC1C2 * SecondRC1C2 * FirstRC1C2;        % Units: (rad)
T_C2_C1     = GenTrMatrix(TrC2_C1);                         % Units: (m)
HomogC1C2   = R_C1_C2/T_C2_C1;                              % Equivalent to: R_C1_C2*inv(T_C2_C1),
                                                            % which is R_C1_C2*T_C1_C2
HomogC2C1   = inv(HomogC1C2);                               % Units: (m) and (rad)
dOC2        = HomogOC1\[TrC2_C1 1]';                        % Equivalent to: HomogC1O*
                                                            % [-paramsC1_C2.pa...

% Homogeneous Transformation, betwheen C0 camera and origin O
HomogC0O =  R_C1_C0*T_C0_C1;

% Understanding Cameras Location and Orientation
figure; title('3D Setup'); grid on; hold on; view(-40,30);
xlabel('X,  (m)'); ylabel('Y,  (m)'); zlabel('Z,  (m)');

Cam1 = plotCamera('Location',dOC1(1:3),'Orientation',HomogOC1(1:3,1:3),'Opacity',0.15,...
                  'AxesVisible',true,'Color','b','Size',CamSize,'Label','C_1');

Cam0 = plotCamera('Location',dOC0(1:3),'Orientation',HomogC1C0(1:3,1:3)*HomogOC1(1:3,1:3),...
                  'Opacity',0.15,'AxesVisible',true,'Color','r','Size',CamSize,'Label','C_0');

Cam2 = plotCamera('Location',dOC2(1:3),'Orientation',HomogC1C2(1:3,1:3)*HomogOC1(1:3,1:3),...
                  'Opacity',0.15,'AxesVisible',true,'Color','g','Size',CamSize,'Label','C_2');
```

```
%% IMAGE REALLYING ========================================================================

% Complete this section if C1 and C2 have different orientation of their coordinates system.
% [JL,JR] = rectifyStereoImages(IL,IR,stereoParams);
JL = IL; JR = IR;



%% GETTING THE DISPARITY MAP ==============================================================

% Compute the map
disptyRnge = [0 256];
disptyMap  = disparity(rgb2gray(IL),rgb2gray(IR),'BlockSize',BlockSize,'DisparityRange',disptyRnge);

% Outlayers removing
disptyMap = outLayersRemoving(disptyMap,tHold);

% 3D Printing of the disparity map
X = 1 : nRows;
Y = 1 : nColu;
[xx,yy] = meshgrid(Y,X);
figure ; mesh(xx,yy,disptyMap);
colorbar; title('3D Disparity Map')
xlabel('xPixels,  (adm)'); ylabel('yPixels,  (adm)'); zlabel('\propto 1/depth');



%% INVERSE PERSPECTIVE GEOMETRY. NEED AT LEAST TWO CAMERAS. NOW WE FIND THE 3D WORLD POINTS OF "F"
%  FROM C1 & C2 POINTS ====================================================================


% Solve with Cramer method. We're now obtainig the World points coordinates with respect to C1
camera by using also C2 datas
alfa = alfa2;                   % Orientation between Camera 1 and Camera 2. From system 1 to system
                                 % 2, along z Camera axis
beta = beta2;                   % Orientation between Camera 1 and Camera 2. From system 1 to system
                                 % 2, along y Camera axis
gama = gama2;                   % Orientation between Camera 1 and Camera 2. From system 1 to system
                                 % 2, along x Camera axis

C2_C1  = HomogOC1\dOC2;         % Vector which crops all C2 components with respect to the C1 Camera
                                 % sys. Equivalent to HomogC1O*dOC2';

XC2_C1 = C2_C1(1);              % First  component
YC2_C1 = C2_C1(2);              % Second component
ZC2_C1 = C2_C1(3);              % Third  component

k1 = conv*(repmat(1:nColu,nRows,1)  -1*nColu/2)/f1;       % Just to semplify the notation
k2 = conv*(repmat(1:nRows,nColu,1)' -1*nRows/2)/f1;       % It goes the same...

k4 = XC2_C1;
k5 = ZC2_C1;
k6 = YC2_C1;

s_a = sin(alfa);    c_a = cos(alfa);
s_b = sin(beta);    c_b = cos(beta);
s_g = sin(gama);    c_g = cos(gama);

j1 = -c_a*c_b;
j2 = -s_a*s_g-c_a*s_b*c_g;
j3 = s_a*c_g-c_a*s_b*s_g;
j4 = -s_a*c_b;
j5 = c_a*s_g-s_a*s_b*c_g;
j6 = -(s_a*s_b*s_g+c_a*c_g);
j7 = s_b;
j8 = -c_b*c_g;
j9 = -c_b*s_g;

k3 = conv*(disptyMap + repmat(1:nColu,nRows,1))/f2;
```

```
% Cramer analytical solution
Detrmnt = (j3*j5-j2*j6) + k3*(j1*j6-j3*j4) + k2*(j3*j8-j2*j9) + k2.*k3*(j1*j9-j3*j7) + ...
           k1*(j5*j9-j6*j8) + k1.*k3*(j6*j7-j4*j9);

XBaseC1 = (-k1*f1*(j2*j6-j3*j5) + k1.*k3*f1*(j1*j6-j3*j4) -k1.*k3*f2*j1*(j5*j9-j6*j8) + ...
            k1.*k3*f2*j2*(j4*j9-j6*j7) -k1.*k3*f2*j3*(j4*j8-j5*j7) + k1*k4*(j5*j9-j6*j8) + ...
           -k1*j2*(k6*j9-k5*j6) + k1*j3*(k6*j8-k5*j5) -k1.*k3*k4*(j4*j9-j6*j7) +...
            k1.*k3*j1*(k6*j9-k5*j6) - k1.*k3*j3*(k6*j7-k5*j4)) ./ Detrmnt;

YBaseC1 = (-k2*f1*(j2*j6-j3*j5) + k2.*k3*f1*(j1*j6-j3*j4) - k2.*k3*f2*j1*(j5*j9-j6*j8) + ...
            k2.*k3*f2*j2*(j4*j9-j6*j7) -k2.*k3*f2*j3*(j4*j8-j5*j7) + k2*k4*(j5*j9-j6*j8)+ ...
           -k2*j2*(k6*j9-k5*j6) + k2*j3*(k6*j8-k5*j5) -k2.*k3*k4*(j4*j9-j6*j7) + ...
            k2.*k3*j1*(k6*j9-k5*j6) - k2.*k3*j3*(k6*j7-k5*j4)) ./ Detrmnt;

ZBaseC1 = (+k3*f2*j1*(j5*j9-j6*j8) - k3*f2*j2*(j4*j9-j6*j7) + k3*f2*j3*(j4*j8-j5*j7) + ...
           -k4*(j5*j9-j6*j8) +j2*(k6*j9-k5*j6) - j3*(k6*j8-k5*j5) + k3*k4*(j4*j9-j6*j7) + ...
           -k3*j1*(k6*j9-k5*j6) +k3*j3*(k6*j7-k5*j4) -k2*f1*(j2*j9-j3*j8) + ...
            k2.*k3*f1*(j1*j9-j3*j7) + k1*f1*(j5*j9-j6*j8) - k1.*k3*f1*(j4*j9-j6*j7)) ./ Detrmnt;



%% REBUILD THE 3D SCENE ===================================================================================

ptsStep   = 1;
sctrStep  = 1;
scatterSize = 2.24;

ptCloud.Location(:,:,1) = XBaseC1;
ptCloud.Location(:,:,2) = YBaseC1;
ptCloud.Location(:,:,3) = ZBaseC1;

ptCloud.Color(:,:,1) = JL(:,:,1);
ptCloud.Color(:,:,2) = JL(:,:,2);
ptCloud.Color(:,:,3) = JL(:,:,3);

points = FromMapToMatrix(ptCloud, ptsStep, 0);

figure
scatter3(XBaseC1(1:sctrStep:end),YBaseC1(1:sctrStep:end),ZBaseC1(1:sctrStep:end),scatterSize,...
        points(1:sctrStep:end,4:6),'fill');
view(-185,-60)
% view(-180,-90)
axis off
grid on
title('RECONSTRUCTED 3D SCENE');



%% GETTING THE IMAGE PLANE 0 IMAGE ========================================================================

% Points coordinates whit respect to the C0 camera system
Cam_coordsX = HomogC0O(1,1)*points(:,1) + HomogC0O(1,2)*points(:,2) + HomogC0O(1,3)*points(:,3) +...
              HomogC0O(1,4)*ones(size(points(:,1)));
Cam_coordsY = HomogC0O(2,1)*points(:,1) + HomogC0O(2,2)*points(:,2) + HomogC0O(2,3)*points(:,3) +...
              HomogC0O(2,4)*ones(size(points(:,1)));
Cam_coordsZ = HomogC0O(3,1)*points(:,1) + HomogC0O(3,2)*points(:,2) + HomogC0O(3,3)*points(:,3) +...
              HomogC0O(3,4)*ones(size(points(:,1)));

[Cam_coordsX, Cam_coordsY, Cam_coordsZ, points] = SortByDist(Cam_coordsX, Cam_coordsY, ...
                                                             Cam_coordsZ, points);

% Coordinates of projected "F points" with respect to image planes
Img_coordsX = f0*Cam_coordsX./(f0-Cam_coordsZ);
Img_coordsY = f0*Cam_coordsY./(f0-Cam_coordsZ);
Img_coordsZ = f0*Cam_coordsZ./(f0-Cam_coordsZ);

figure
scatter(Img_coordsX(1:sctrStep:end),Img_coordsY(1:sctrStep:end),scatterSize, ...
        points(1:sctrStep:end,4:6),'fill');
view(180,-90)
axis off
title('Synthesized image on C_0 image plane');
```

```
%% END PROGRAM_ ============================================================================
```

9.1.8 – outLayersRemoving.m

```
function [ Filtered_Mat ] = outLayersRemoving( Mat, tHold )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

    infti      = find(Mat < -1e30 | Mat > 1e30);
    Mat(infti) = nan;

    survivors = find(Mat >= -1e30 & Mat <= 1e30);

    nn = numel(survivors);

    media  = (sum(sum(Mat(survivors))))/nn;
    sigma  = sqrt(1/(nn-1) * sum(sum((Mat(survivors)-media).^2)));

    outLrs = find(Mat > media+tHold*sigma | Mat < media-tHold*sigma);
    Mat(outLrs) = nan;

    Filtered_Mat = Mat;

end
```

9.1.9 – FromMapToMatrix.m

```
function [ points ] = FromMapToMatrix(ptCloud, ptsStep, flag)
%FromMapToMatrix transforms points coordnates from a MAP form to
%nx6 Matrix,where n is the total number of pixels of the image "Image".
%   ptCloud is a pointCloud object determined by using reconstructScene
%   and ptsStep is the step between two consecutive points. Increase this
%   number to get a faster run of the program, but quality may goes down.

tic
XBase0 = ptCloud.Location(1:ptsStep:end,1:ptsStep:end,1);
YBase0 = ptCloud.Location(1:ptsStep:end,1:ptsStep:end,2);
ZBase0 = ptCloud.Location(1:ptsStep:end,1:ptsStep:end,3);
if flag == 1
    fprintf('Reducing points density.-- Elapsed time: %.2f sec\n',toc);
end

tic
aux_c_R = ptCloud.Color(1:ptsStep:end,1:ptsStep:end,1);
cR = single(aux_c_R(:));

aux_c_G = ptCloud.Color(1:ptsStep:end,1:ptsStep:end,2);
cG = single(aux_c_G(:));

aux_c_B = ptCloud.Color(1:ptsStep:end,1:ptsStep:end,3);
cB = single(aux_c_B(:));

c = [cR/255, cG/255, cB/255];

points = [XBase0(:), YBase0(:), ZBase0(:), c];
if flag == 1
    fprintf('From map form to nx6.----- Elapsed time: %.2f sec\n',toc);
end

end
```

9.1.10 – SortByDist.m

```
function [ X_ord, Y_ord, Z_ord, points_ord ] = SortByDist( X, Y, Z, points)
%SortByDist Reord points by distance
%   X is a vector which crops all x components of the points
%   Y is a vector which crops all y components of the points
%   Z is a vector which crops all z components of the points
%   points is an object needed to take into account pixels color

dist  = sqrt(X.^2 + Y.^2 + Z.^2);
[~,idx]     = sort(dist,'descend');
points_ord  = points(idx,:);
X_ord = X(idx,:);
Y_ord = Y(idx,:);
Z_ord = Z(idx,:);

end
```

9.1.11 – DisparityAlghorithm.m

```
%% SAD ALGORITHM

% INITIALIZATION
clear;
clc;
close all



%% DATA

f = .001;       % Cameras focal length, (m)
baseLine = .1;  % Distance between the cameras, (m)

percntg = .25;  % Proportional to the searching length

I_L = imread('left.jpg');    % figure();  imshow(I_L);
I_R = imread('right.jpg');   % figure();  imshow(I_R);

I_L = rgb2gray(I_L);         % figure();  imshow(I_L);
I_R = rgb2gray(I_R);         % figure();  imshow(I_R);

nRows = numel(I_L(:,1));
nColu = numel(I_L(1,:));

figure('units','normalized','outerposition',[0 0 1 1])                    % Full screen figure
subplot(1,2,1); imshow(I_L); hold on; title('Left image loaded: I_L')
subplot(1,2,2); imshow(I_R); hold on; title('Right image loaded: I_R')

I_L = single(I_L);  % Single and not double in order to reduce the time processing
I_R = single(I_R);  % It goes the same...


%% SELECTION ALGORITHM

a = 5;  % [<--a-->  pivot  <--a-->]
b = 4;  % Similarly, but vertically

% Respectively: col = x ; row = nRows +1 -y
% row = 302;
% col = 132;
% P_L_Img = [col row];            % Point of the left image in Images coordinates
% P_L_Mtb = [col nRows-(row-1)];  % Point of the left image in Matlab coordinates
%
% figure();   % Understanding the different coordinates systems
% plot(P_L_Img(1),P_L_Img(2),'ro','MarkerSize',5); grid on; hold on
% plot(P_L_Mtb(1),P_L_Mtb(2),'bo','MarkerSize',5);
% return

MaxDist = ceil(percntg*nColu/2);

% SSD alghorithm
% Preallocating
CorrelMat = zeros(nRows-b,nColu-a);
DisptyMat = zeros(nRows-b,nColu-a);
tic;
advnce = [0   (nRows-2*b)*(nColu-2*a)];
```

```
for IdxRow = 1 : 1 : nRows

    for IdxCol = 1 : 1 : nColu

        P_L_Img = [IdxCol IdxRow];                    % Point of the left image in Images
                                                      % coordinates
        P_L_Mtb = [P_L_Img(1) nRows-(P_L_Img(2)-1)];  % Point of the left image in Matlab
                                                      % coordinates

        if P_L_Mtb(1)-a >=  1   &&  P_L_Mtb(1)+a <= nColu  &&...     % All windows points fall into
                                                                     % the image pixels
           P_L_Mtb(2)-b >=  1   &&  P_L_Mtb(2)+b <= nRows

            advnce = [advnce(1)+1  advnce(2)];

            subplot(1,2,1); L = scatter(IdxCol,IdxRow,10,'ro','LineWidth',1.5);

            tt = advnce(1)/advnce(2)*100;
            fprintf('Processing element %d/%d... Percntge = %.2f %%\n ', advnce(1),advnce(2),tt);

            leftNode = IdxCol-MaxDist;
            righNode = IdxCol+   0   ;

            if leftNode < 1+a
                leftNode = a+1;       % disp('qui')
            end

            if righNode > nColu-a
                righNode = nColu-a-1; % disp('qua')
            end

            ll = 1;
            SAD = zeros(1,numel(leftNode:righNode));
            for kk = leftNode : 1 : righNode

               subplot(1,2,2); printMask(a,a,b,b,IdxRow,kk);  % Input order: al,ar, bd,bu, #rows,
                                                              % #column

                LeftMat = I_L(IdxRow-b:IdxRow+b,IdxCol-a:IdxCol+a);
                RighMat = I_R(IdxRow-b:IdxRow+b,    kk-a:    kk+a);

                Parzial = sum((LeftMat - RighMat).^2);
                SAD(ll) = sum(Parzial);

                ll = ll + 1;

            end
            [Val_min , Idx_min] = min(SAD);
            Idx_min = Idx_min + leftNode-1;

            CorrelMat(IdxRow,IdxCol,1) = Idx_min;
            CorrelMat(IdxRow,IdxCol,2) = Val_min;

            DisptyMat(IdxRow,IdxCol) = IdxCol - Idx_min;

           subplot(1,2,1); delete(L)

        end

    end

end

disp(' '); toc;
```

```
%% GETTING THE DISPARITY MAP

% Depth map
Z = f*baseLine./DisptyMat;

figure()
imshow(DisptyMat(:,:,1), [0, 256]);
title('Disparity map')
colormap jet
colorbar

% 3D Depth map
[x,y] = size(DisptyMat(:,:,1));
X = 1 : x;
Y = 1 : y;
[xx,yy] = meshgrid(Y,X);
i = im2double(DisptyMat(:,:,1));
figure ; mesh(xx,yy,i);
colorbar



%% Outlayers remuving

% Filtered Depth map
tHold = 4;

media = (sum(sum(CorrelMat(:,:,2))))/numel(CorrelMat(:,:,2));
sigma = sqrt(1/((numel(CorrelMat(:,:,2))-1)) * sum(sum((CorrelMat(:,:,2)-media).^2)));
[rowsD,coluD] = find(CorrelMat(:,:,2) < media-tHold*sigma);
[rowsU,coluU] = find(CorrelMat(:,:,2) > media+tHold*sigma);

CorrelMat_ = CorrelMat;
DisptyMat_ = DisptyMat;

CorrelMat_(rowsD,coluD,2) = nan;
DisptyMat_(rowsU,coluU,2) = nan;

DisptyMat_(rowsD,coluD) = nan;
DisptyMat_(rowsU,coluU) = nan;


% 3D Filtered Depth map
[x,y] = size(DisptyMat_(:,:,1));
X = 1 : x;
Y = 1 : y;
[xx,yy] = meshgrid(Y,X);
i = im2double(DisptyMat_(:,:,1));
figure ; mesh(xx,yy,i);
colorbar


%% _END PROGRAM_
```

9.1.12 – Calibrazione_C1_C0.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%         STEREO CAMERA CALIBRATION              %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        Michele Pecchio Ghiringhelli Rota       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%             Last update: 16/01/2019            %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALIZATION ===================================================================================

clear;
clc;
close all

CurrentFlder = pwd;



%% IMAGE LOADING ===================================================================================

% Preparing left images
D_C0 = [CurrentFlder '\C0'];
S = dir(fullfile(D_C0,'WP*.jpg')); % pattern to match filenames.
for k = 1:numel(S)

    F = fullfile(D_C0,S(k).name);
    I = imread(F);
    I = imresize(I,3264/3072);
    imshow(I)
    baseFileName = ['I' num2str(k) '_Ready.jpg'];
    fullFileName = fullfile([CurrentFlder '\C0\Ready'], baseFileName);
    imwrite(I, fullFileName);

end
D_C0 = [CurrentFlder '\C0\Ready'];

% Right images
D_C1 = [CurrentFlder '\C1\Ready'];



%% SPECIFY CALIBRATION IMAGES ======================================================================

leftImages = imageSet(D_C0);
imageFileNamesLeft  = leftImages.ImageLocation;
celldisp(imageFileNamesLeft);

rightImages = imageSet(D_C1);
imageFileNamesRight = rightImages.ImageLocation;
celldisp(imageFileNamesRight);

figure(1)
subplot(2,1,1)
montage(imageFileNamesLeft, 'Size',[1 length(S)]);
title('Images taken from C_0 Camera')

subplot(2,1,2)
montage(imageFileNamesRight,'Size',[1 length(S)]);
title('Images taken from C_1 Camera')
```

```
%% DETECT THE CHECKERBOARDS ================================================================
[imagePoints, boardSize] = detectCheckerboardPoints(rightImages.ImageLocation,...
                                                    leftImages.ImageLocation);



%% PLOT DETECTED POINTS =====================================================================

% Left images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    imshow(imageFileNamesLeft{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    hold on
    plot(imagePoints(:, 1, ii, 2), imagePoints(:, 2, ii, 2), 'go');
    hold on
end

% Right images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    imshow(imageFileNamesRight{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    hold on
    plot(imagePoints(:, 1, ii, 1), imagePoints(:, 2, ii, 1), 'go');
    hold off
end



%% MANUALLY REWRITING OF THE WRONG VALUES ===================================================
%{
imagePoints(:,:,2,1) = [];

imagePoints(:,:,1,2) = [];

figure();

subplot(2,5,2)
hold on
plot(imagePoints(:, 1, 2, 1), imagePoints(:, 2, 2, 1), 'go');

subplot(2,5,6)
hold on
plot(imagePoints(:, 1, 1, 2), imagePoints(:, 2, 1, 2), 'go');

%}
```

```
%% ESTIMATION OF THE PARAMETERS ===========================================================

% Specify world coordinates of checkerboard keypoints.
squareSize = 100; % in millimeters
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the stereo camera system.
params = estimateCameraParameters(imagePoints, worldPoints);

% Visualize calibration accuracy.
figure;
showReprojectionErrors(params);

% Visualize camera extrinsics.
figure;
showExtrinsics(params);

% Plot detected and reprojected points.
figure;
imshow(imageFileNamesRight{1});
title('Detected VS Reprojected Points')
hold on
plot(imagePoints(:, 1, 1, 1), imagePoints(:, 2, 1, 1), 'go');
plot(params.CameraParameters1.ReprojectedPoints(:, 1, 1, 1),
params.CameraParameters1.ReprojectedPoints(:, 2, 1, 1), 'r+');
legend('Detected Points', 'ReprojectedPoints');
hold off

save C1_C0_Params.mat params



%% END PROGRAM_ ============================================================================
```

9.1.13 – Calibrazione_C1_C2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        STEREO CAMERA CALIBRATION               %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%       Michele Pecchio Ghiringhelli Rota        %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            Last update: 16/01/2019             %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALIZATION ==================================================================================

clear;
clc;
close all

CurrentFlder = pwd;



%% IMAGE LOADING ===================================================================================

% Preparing left images
D_C1 = [CurrentFlder '\C1'];
S = dir(fullfile(D_C1,'2019*.jpg')); % pattern to match filenames.
for k = 1:numel(S)

    F = fullfile(D_C1,S(k).name);
    I = imread(F);
    I = imrotate(I,-90);
%     imshow(I)
    baseFileName = ['I' num2str(k) '_Ready.jpg'];
    fullFileName = fullfile([CurrentFlder '\C1\Ready'], baseFileName);
    imwrite(I, fullFileName);

end
D_C1 = [CurrentFlder '\C1\Ready'];

% Preparing right images
D_C2 = [CurrentFlder '\C2'];
S = dir(fullfile(D_C2,'2019*.jpg')); % pattern to match filenames.
for k = 1:numel(S)

    F = fullfile(D_C2,S(k).name);
    I = imread(F);
    I = imrotate(I,-90);
%     imshow(I)
    baseFileName = ['I' num2str(k) '_Ready.jpg'];
    fullFileName = fullfile([CurrentFlder '\C2\Ready'], baseFileName);
    imwrite(I, fullFileName);

end
D_C2 = [CurrentFlder '\C2\Ready'];



%% SPECIFY CALIBRATION IMAGES ======================================================================

leftImages = imageSet(D_C1);
imageFileNamesLeft  = leftImages.ImageLocation;
celldisp(imageFileNamesLeft);

rightImages = imageSet(D_C2);
imageFileNamesRight = rightImages.ImageLocation;
celldisp(imageFileNamesRight);

figure(1)
subplot(2,1,1)
montage(imageFileNamesLeft, 'Size',[1 length(S)]);
title('Images taken from C_1 Camera')

subplot(2,1,2)
montage(imageFileNamesRight,'Size',[1 length(S)]);
title('Images taken from C_2 Camera')
```

```
%% DETECT THE CHECKERBOARDS ================================================================================
[imagePoints, boardSize] = detectCheckerboardPoints(leftImages.ImageLocation,...
                                                    rightImages.ImageLocation);


%% PLOT DETECTED POINTS ====================================================================================
% Left images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    imshow(imageFileNamesLeft{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    hold on
    plot(imagePoints(:, 1, ii, 1), imagePoints(:, 2, ii, 1), 'go');
    hold on
end

% Right images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    imshow(imageFileNamesRight{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    hold on
    plot(imagePoints(:, 1, ii, 2), imagePoints(:, 2, ii, 2), 'go');
    hold off
end


%% MANUALLY REWRITING OF THE WRONG VALUES ==================================================================
%{
imagePoints(:,:,2,1) = [];

imagePoints(:,:,1,2) = [];

figure();

subplot(2,5,2)
hold on
plot(imagePoints(:, 1, 2, 1), imagePoints(:, 2, 2, 1), 'go');

subplot(2,5,6)
hold on
plot(imagePoints(:, 1, 1, 2), imagePoints(:, 2, 1, 2), 'go');

%}
```

```
%% ESTIMATION OF THE PARAMETERS ===========================================================

% Specify world coordinates of checkerboard keypoints.
squareSize = 100; % in millimeters
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the stereo camera system.
params = estimateCameraParameters(imagePoints, worldPoints);

% Visualize calibration accuracy.
figure;
showReprojectionErrors(params);

% Visualize camera extrinsics.
figure;
showExtrinsics(params);

% Plot detected and reprojected points.
figure;
imshow(imageFileNamesLeft{1});
title('Detected VS Reprojected Points')
hold on
plot(imagePoints(:, 1, 1, 1), imagePoints(:, 2, 1, 1), 'go');
plot(params.CameraParameters1.ReprojectedPoints(:, 1, 1, 1),
params.CameraParameters1.ReprojectedPoints(:, 2, 1, 1), 'r+');
legend('Detected Points', 'ReprojectedPoints');
hold off

save C1_C2_Params.mat params


%% END PROGRAM_ ===========================================================================
```

9.1.14 – Script_finaleNoRealTImeNoEyesDetection.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%         IMAGE REPROJECTION ON USER CAM         %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%       Michele Pecchio Ghiringhelli Rota        %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            Last update: 26/01/2019             %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALIZATION ===================================================================================

clear
clc
close all

addpath(genpath('C0'));
addpath(genpath('C1'));
addpath(genpath('C2'));

paramsC1_C2 = load('C1_C2_Params.mat'); % Stereo parameters of C1 and C2
paramsC1_C0 = load('C1_C0_Params.mat'); % Stereo parameters of C1 and C0



%% DATA & SETTINGS =================================================================================

% Time counter
disp('%%%% TIME FROM IMAGES LOADING TO REPROJECTION %%%%')

% Images loading
tic
IL = imresize(imrotate(imread('Da_C1_A.jpg'),1*-90),1); % figure; imshow(IL);
IR = imresize(imrotate(imread('Da_C2_A.jpg'),1*-90),1); % figure; imshow(IR);
fprintf('Images loading.----------- Elapsed time: %.2f sec\n',toc);

% Disparity settings
BlockSize    = 11;                          % Size of the matching windows,
                                            % (pixels)
BlockSizeVec = [7 13 17];                   % Size of the matching windows (for following test),
                                            % (pixels)
DisptyRnge   = [0 128];                     % [minDisparity maxDisparity],
                                            % (pixel)
UniqssTHold  = 8;                           % Increasing to marking more pixels as unreliable. 15
                                            % (default)
TextrThhold  = 2e-8;                        % Minimum texture threshold. Lower values cause less
                                            % reliable disptyMap. 2e-4 (default)

% Reduce the elapsed time by increasing the following parameters. Quality may decrease
ptsStep  = 1;                               % Step of the 3D points obtained
sctrStep = 10;                              % Step of the scatter-reconstructed figure.

% Others data
scatterSize = 2.24;                         % Proportional to the area of the pixel on
mOverpixel  = 1;                            % Length (m) per pixels,           (m/pixel)
CamSize     = .05;                          % Size of the Cameras on the plot, (m)
Hor         = .37;                          % To horizontally shift the matching figure
Ver         = .077;                         % To vertically   shift the matching figure

% Camera C1 Data:
dOC1  = [0 0 0];                            % Coordinates of the camera C1,               (m)
alpha = 0;                                  % Rotation angle around the original X axis, (°)

% Camera C0 Data:
f0 = 35/1000*mOverpixel;                    % Camera C0 focal length,                     (m)
```

```
%% PRELIMINARY CALCULATIONS ================================================================================

% Homogeneous Transformation, betwheen C1 camera and O origin
RxMatrix = rotx(alpha);
RxMATRIX = [RxMatrix [0 0 0]'; [0 0 0 1]];
HomogOC1 = RxMATRIX;
HomogC1O = inv(HomogOC1);

% Homogeneous Transformation, betwheen C1 camera and C0 camera
R_C1_C0 = [paramsC1_C0.params.RotationOfCamera2' [0 0 0]'; [0 0 0 1]];  % Units: (rad)
T_C0_C1 = GenTrMatrix(paramsC1_C0.params.TranslationOfCamera2/1000);    % Units: (m)
HomogC1C0 = R_C1_C0/T_C0_C1;                                            % Equivalent to:
                                                                        % R_C1_C0*inv(T_C0_C1),
                                                                        % which is R_C1_C0*T_C1_C0
HomogC0C1 = inv(HomogC1C0);                                             % Units: (m) and (rad)
dOC0 = HomogOC1\[-paramsC1_C0.params.TranslationOfCamera2/1000 1]';     % Equivalent to: HomogC1O*[-
                                                                        % paramsC1_C0.pa...

% Homogeneous Transformation, betwheen C1 camera and C2 camera
R_C1_C2 = [paramsC1_C2.params.RotationOfCamera2' [0 0 0]'; [0 0 0 1]];  % Units: (rad)
T_C2_C1 = GenTrMatrix(paramsC1_C2.params.TranslationOfCamera2/1000);    % Units: (m)
HomogC1C2 = R_C1_C2/T_C2_C1;                                            % Equivalent to:
                                                                        % R_C1_C2*inv(T_C2_C1),
                                                                        % which is R_C1_C2*T_C1_C2
HomogC2C1 = inv(HomogC1C2);                                             % Units: (m) and (rad)
dOC2 = HomogOC1\[-paramsC1_C2.params.TranslationOfCamera2/1000 1]';     % Equivalent to: HomogC1O*
                                                                        % [-paramsC1_C2.pa...

% Homogeneous Transformation, betwheen C0 camera and origin O
HomogC0O =  R_C1_C0*GenTrMatrix(paramsC1_C0.params.TranslationOfCamera2/1000);

% Understanding Cameras Location and Orientation
figure; title('3D Setup'); grid on; hold on; view(-40,30)
xlabel('X,  (m)'); ylabel('Y,  (m)'); zlabel('Z,  (m)');

Cam1 = plotCamera('Location',[0 0 0],'Orientation',HomogOC1(1:3,1:3), ...
                  'Opacity',0.15,'AxesVisible',true,'Color','b','Size',CamSize,'Label','C_1');

Cam0 = plotCamera('Location',dOC0(1:3),'Orientation',HomogC1C0(1:3,1:3)*HomogOC1(1:3,1:3), ...
                  'Opacity',0.15,'AxesVisible',true,'Color','r','Size',CamSize,'Label','C_0');

Cam2 = plotCamera('Location',dOC2(1:3),'Orientation',HomogC1C2(1:3,1:3)*HomogOC1(1:3,1:3), ...
                  'Opacity',0.15,'AxesVisible',true,'Color','g','Size',CamSize,'Label','C_2');



%% IMAGE REALLYING ========================================================================================

figure
subplot(1,2,1); imshow(IL)
title('Captured scene by C_1')
subplot(1,2,2); imshow(IR)
title('Captured scene by C_2')

stereoParams = paramsC1_C2.params;
tic
[JL, JR] = rectifyStereoImages(IL,IR,stereoParams);
fprintf('Images rectification.----- Elapsed time: %.2f sec\n',toc);

figure
subplot(1,2,1); imshow(JL)
title('Realigned image from C_1')
subplot(1,2,2); imshow(JR)
title('Realigned image from C_2')
```

```
%% GETTING THE DISPARITY MAP ==============================================================

% Compute the map
tic
JL_bw = rgb2gray(JL);
JR_bw = rgb2gray(JR);
disptyMap  = disparity(JL_bw,JR_bw,'BlockSize',BlockSize,'DisparityRange',DisptyRnge, ...
                       'UniquenessThreshold',UniqssTHold);
fprintf('Disparity computation.---- Elapsed time: %.2f sec\n',toc);
Nout = length(disptyMap(:) <= -3.402e+38);   % It varies along the BlockSize

figure;
imshow(disptyMap, [0, 256]);
warning on verbose
title(['Disparity Map. BlockSize = ' num2str(BlockSize)]);
colormap jet; colorbar



%% INVERSE PERSPECTIVE GEOMETRY. NEED AT LEAST TWO CAMERAS. NOW WE FIND THE 3D WORLD POINTS OF "F"
%  FROM C1 & C2 POINTS ====================================================================

tic
xyzPoints = reconstructScene(disptyMap,stereoParams);
xyzPoints = xyzPoints./1000;    % Convert to meter
fprintf('Getting world points.----- Elapsed time: %.2f sec\n',toc);

% Create a pointCloud object
ptCloud = pointCloud(xyzPoints, 'Color', JL);

% Create a streaming point cloud viewer
player3D = pcplayer([-2.5, 2.5], [-4, 1], [0.5, 8], 'VerticalAxis', 'y', ...
                    'VerticalAxisDir', 'down');
% Visualize the point cloud
view(player3D, ptCloud);



%% SYNTHESIZING OF THE IMAGE PLANE C0 IMAGE ===============================================

points = FromMapToMatrix(ptCloud, ptsStep, 1);

% Points coordinates whit respect to the C0 camera system
tic
Cam_coordsX = HomogC0O(1,1)*points(:,1) + HomogC0O(1,2)*points(:,2) + HomogC0O(1,3)*points(:,3) +...
              HomogC0O(1,4)*ones(size(points(:,1)));
Cam_coordsY = HomogC0O(2,1)*points(:,1) + HomogC0O(2,2)*points(:,2) + HomogC0O(2,3)*points(:,3) +...
              HomogC0O(2,4)*ones(size(points(:,1)));
Cam_coordsZ = HomogC0O(3,1)*points(:,1) + HomogC0O(3,2)*points(:,2) + HomogC0O(3,3)*points(:,3) +...
              HomogC0O(3,4)*ones(size(points(:,1)));
tPart = toc;

% Reoder points from the closest to the farthest, in order to cover the hidden image points
tic
[Cam_coordsX, Cam_coordsY, Cam_coordsZ, points] = SortByDist(Cam_coordsX, Cam_coordsY,...
                                                             Cam_coordsZ, points);
fprintf('Sorting pts by distance.-- Elapsed time: %.2f sec\n',toc);

% Outlayers removing. It can be also useful to remove object over or less than a certain treshold
tic
outLay = Cam_coordsX(:) < -2.5 | Cam_coordsX(:) > 2.5;
Cam_coordsX(outLay) = NaN;
outLay = Cam_coordsY(:) < -5   | Cam_coordsY(:) > 1;
Cam_coordsY(outLay) = NaN;
outLay = Cam_coordsZ(:) < .3   | Cam_coordsZ(:) > 8;
Cam_coordsZ(outLay) = NaN;
fprintf('Filtering outlayers.------ Elapsed time: %.2f sec\n',toc);
```

```
% Hidden points removing. (To be completed if necessary...)
%{

Theta = atan(Cam_coordsZ./Cam_coordsX);
phi   = atan(Cam_coordsY./Cam_coordsZ);

minVal = .000005;
maxVal = .000005;

[rows,~] =   find( (Theta-minVal <= Theta & Theta <= Theta+maxVal)  &...
                   (phi-  minVal <= phi   & phi   <= phi+  maxVal));

%...

%}

% Coordinates of projected points with respect to image planes C0. Pinhole camera model
tic
Img_coordsX = f0*Cam_coordsX./(f0-Cam_coordsZ);
Img_coordsY = f0*Cam_coordsY./(f0-Cam_coordsZ);
Img_coordsZ = f0*Cam_coordsZ./(f0-Cam_coordsZ);
fprintf('Perspective transform.---- Elapsed time: %.2f sec\n',tPart+toc);

figure
scatter(Img_coordsX(1:sctrStep:end),Img_coordsY(1:sctrStep:end),scatterSize, ...
        points(1:sctrStep:end,4:6),'fill');
fprintf('Time to pixels on.-------- Elapsed time: -ND- sec\n\n\n');
view(180,-90)
axis off
title('Synthesized image on C_0 image plane');
return
```

```
%% IMPROVING DISPARITY MAP, 3D AND SYNTHESIZED IMAGE. IT NEEDS MORE TIME... ========================

disp('%%%% TIME FOR PTS COMPUT. INPUT=(JL_bw,JR_bw) %%%%')
for ii = 1 : length(BlockSizeVec)

    tic;
    disptyMap  = disparity(JL_bw,JR_bw,'BlockSize',BlockSizeVec(ii),'DisparityRange',DisptyRnge, ...
                           'UniquenessThreshold',UniqssTHold);

    xyzPoints = reconstructScene(disptyMap,stereoParams);
    xyzPoints = xyzPoints./1000;     % Convert to meter

    points = FromMapToMatrix(ptCloud, ptsStep, 0);

    % Points coordinates whit respect to the C0 camera system
    Cam_coordsX = HomogC0O(1,1)*points(:,1) + HomogC0O(1,2)*points(:,2) +...
                  HomogC0O(1,3)*points(:,3) + HomogC0O(1,4)*ones(size(points(:,1)));
    Cam_coordsY = HomogC0O(2,1)*points(:,1) + HomogC0O(2,2)*points(:,2) +...
                  HomogC0O(2,3)*points(:,3) + HomogC0O(2,4)*ones(size(points(:,1)));
    Cam_coordsZ = HomogC0O(3,1)*points(:,1) + HomogC0O(3,2)*points(:,2) +...
                  HomogC0O(3,3)*points(:,3) + HomogC0O(3,4)*ones(size(points(:,1)));

    % Reoder points from the closest to the farthest, in order to cover the hidden image points
    [Cam_coordsX, Cam_coordsY, Cam_coordsZ, points] = SortByDist(Cam_coordsX, Cam_coordsY,...
                                                                 Cam_coordsZ, points);

    % Outlayers removing. It can be also useful to remove object over or less than a certain
    % treshold
    outLay = Cam_coordsX(:) < -2.5 | Cam_coordsX(:) > 2.5;
    Cam_coordsX(outLay) = NaN;
    outLay = Cam_coordsY(:) < -5   | Cam_coordsY(:) > 1;
    Cam_coordsY(outLay) = NaN;
    outLay = Cam_coordsZ(:) < .3   | Cam_coordsZ(:) > 8;
    Cam_coordsZ(outLay) = NaN;

    % Coordinates of projected points with respect to image planes C0. Pinhole camera model
    Img_coordsX = f0*Cam_coordsX./(f0-Cam_coordsZ);
    Img_coordsY = f0*Cam_coordsY./(f0-Cam_coordsZ);
    Img_coordsZ = f0*Cam_coordsZ./(f0-Cam_coordsZ);

    fprintf('Time over one cicle (without display)  : %.2f sec\n',toc);

    % Several printing of the images on plane C0
    figure(7);
    scatter(Img_coordsX(1:sctrStep:end),Img_coordsY(1:sctrStep:end),scatterSize,...
    points(1:sctrStep:end,4:6),'fill');
    hold on
    view(180,-90)
    axis off
    title(['Synthesized image on C_0. Overlapped Disparity map = ' num2str(ii)])

    figure(8);
    subplot(1,2,1)
    imshow(imresize(imrotate(imread('Da_C0_A.jpg'),-90),.65))
    axis image
    title('Captured image from C_0')

    subplot(1,2,2)
    scatter(Img_coordsX(1:sctrStep:end),Img_coordsY(1:sctrStep:end),scatterSize,...
    points(1:sctrStep:end,4:6),'fill');
    title('Synthesized image on C_0')
    hold on
    axis image
    V = axis;
    axis([V(1)+Hor*V(1)  V(2)+Hor*V(1)  ,  V(3)+1.1*Ver*V(3)  V(4)+Ver*V(3)])
    view(180,-90)
    axis off

end
fprintf('\n\n')


%% END PROGRAM_=============================================================================================
```

9.1.15 – ImageCapture.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            WEBCAM IMAGES ACQUISITOR             %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        Michele Pecchio Ghiringhelli Rota       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%             Last update: 11/02/2019            %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% INITIALIZATION ===================================================================================

% Remember to check if the Cameras names are well setted:
% to remind the user what will not be updated,
% is marked with "WARNING!!" the lines in question.

clear
clc
close all
addpath(genpath('Function folder'))


%% DATA SETTINGS ===================================================================================
nColu = 320;
nRows = 240;
Resoultn = [num2str(nColu) 'x' num2str(nRows)]; % Resolution of each image

camList  = webcamlist
nCameras = length(camList);

C1 = webcam(3);                    % WARNING!!
C0 = webcam(1);                    % WARNING!!
C2 = webcam(4);                    % WARNING!!
% preview(C2)
% return

C0.resolution = Resoultn;
C1.resolution = Resoultn;
C2.resolution = Resoultn;

DATA_Struct(1).CamName = 'C0';     % WARNING!!
DATA_Struct(2).CamName = 'C1';     % WARNING!!
DATA_Struct(3).CamName = 'C2';     % WARNING!!

DATA_Struct(1).Resolution = Resoultn;
DATA_Struct(2).Resolution = Resoultn;
DATA_Struct(3).Resolution = Resoultn;

mkdir C0                           % WARNING!!
mkdir C1                           % WARNING!!
mkdir C2                           % WARNING!!

CurrentFolder = pwd;
saveDirC0 = [CurrentFolder '\C0']; % WARNING!!
saveDirC1 = [CurrentFolder '\C1']; % WARNING!!
saveDirC2 = [CurrentFolder '\C2']; % WARNING!!
```

```
%% INTERACTION WITH THE USER ==========================================================================

disp('Digitare il numero di immagini che si intende acquisire e premere invio.')
nPhotos = str2double(input('Digitare numero: ','s'));
disp(' ')

disp('Si desidera visualizzare in anteprima il rilevamento della CheckerBoard?')
disp('Si digiti la lettera "y" per confermare o "n" per smentire.')
CheckerB = input('Digitare scelta: ','s');
disp(' ')
if strcmp(CheckerB,'y') == 1
    disp('SI PRESTI ATTENZIONE AL RITARDO DI ACQUISIZIONE !!')
    pause(4)
end

disp(' ')

% Global variable definition to get an interactive figure
global KEY_IS_PRESSED
KEY_IS_PRESSED = 0;

figure('units','normalized','outerposition',[0 0 1 1])
gcf;
set(gcf, 'KeyPressFcn', @myKeyPressFcn)

% Interactive loop tho takes the photos
for j = 1 : 1 : nPhotos

    disp(' ')

    disp   ('Posizionare la CheckBoard e digitare invio sulla figura 1')
    fprintf('per acquisire il %2d° set di immagini:\n\n',j);

    while ~KEY_IS_PRESSED % Stops the cycle of a necessary time
        drawnow

        subplot(1,3,1); I0 = snapshot(C0); image(I0); title('Hidden observer Camera  (C_0)')
        drawnow
        if strcmp(CheckerB,'y') == 1
            hold on
            [imagePoints, ~] = detectCheckerboardPoints(I0);
            if isempty(imagePoints) == 0;
                plot(imagePoints(:, 1, 1), imagePoints(:, 2, 1), 'ro','MarkerFaceColor','r');
                drawnow; clear imagePoints
            end
            hold off
        end

        subplot(1,3,2); I1 = snapshot(C1); image(I1); title('Left  Camera  (C_1)')
        drawnow
        if strcmp(CheckerB,'y') == 1
            hold on
            [imagePoints, ~] = detectCheckerboardPoints(I1);
            if isempty(imagePoints) == 0;
                plot(imagePoints(:, 1, 1), imagePoints(:, 2, 1), 'ro','MarkerFaceColor','r');
                drawnow; clear imagePoints
            end
            hold off
        end


        subplot(1,3,3); I2 = snapshot(C2); image(I2); title('Right Camera  (C_2)')
        drawnow
        if strcmp(CheckerB,'y') == 1
            hold on
            [imagePoints, ~] = detectCheckerboardPoints(I2);
            if isempty(imagePoints) == 0;
                plot(imagePoints(:, 1, 1), imagePoints(:, 2, 1), 'ro','MarkerFaceColor','r');
                drawnow; clear imagePoints
            end
            hold off
        end
    end
```

```
    genvarname( 'DATA_Struct(1).Img',num2str(j));
    eval      (['DATA_Struct(1).Img' num2str(j) '= snapshot(C0)']); % WARNING!!

    genvarname( 'DATA_Struct(2).Img',num2str(j));
    eval      (['DATA_Struct(2).Img' num2str(j) '= snapshot(C1)']); % WARNING!!

    genvarname( 'DATA_Struct(3).Img',num2str(j));
    eval      (['DATA_Struct(3).Img' num2str(j) '= snapshot(C2)']); % WARNING!!

    StructFields = fieldnames(DATA_Struct);
    ImageName    = StructFields{2+j};

    % Saving j^th C0 photo
    fotoC0 = getfield(DATA_Struct(1),ImageName);       % WARNING!!
    baseFileName = sprintf('Img%02d_C0.png', j);      % WARNING!!
    fullFileName = fullfile(saveDirC0, baseFileName);
    imwrite(fotoC0, fullFileName);

    % Saving j^th C1 photo
    fotoC1 = getfield(DATA_Struct(2),ImageName);       % WARNING!!
    baseFileName = sprintf('Img%02d_C1.png', j);       % WARNING!!
    fullFileName = fullfile(saveDirC1, baseFileName);
    imwrite(fotoC1, fullFileName);

    % Saving j^th C2 photo
    fotoC2 = getfield(DATA_Struct(3),ImageName);       % WARNING!!
    baseFileName = sprintf('Img%02d_C2.png', j);       % WARNING!!
    fullFileName = fullfile(saveDirC2, baseFileName);
    imwrite(fotoC2, fullFileName);

    % Warn the user
    subplot(1,3,1); ax = gca; ax.FontSize = 2*ax.FontSize; title('IMAGE CAPTURED','Color','r');
    subplot(1,3,2); ax = gca; ax.FontSize = 2*ax.FontSize; title('IMAGE CAPTURED','Color','r');
    subplot(1,3,3); ax = gca; ax.FontSize = 2*ax.FontSize; title('IMAGE CAPTURED','Color','r');
    pause(1)

    KEY_IS_PRESSED = 0;

end

KEY_IS_PRESSED = 1;
clear global

save Images_SET DATA_Struct

fprintf('\n\nEsecuzione terminata\n\n');
close all

% Partorito = load('Images_SET.mat');

%% END_PROGRAM ========================================================================================
```

9.1.16 – myKeyPressFcn.m

```
function myKeyPressFcn(hObject, event)
global KEY_IS_PRESSED
KEY_IS_PRESSED  = 1;
disp('Key is pressed. The photos were taken')
end
```

9.1.17 – Calibrazione_C1_C0.m – Con settaggio parametri del modello per calibrare

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            STEREO CAMERA CALIBRATION           %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        Michele Pecchio Ghiringhelli Rota       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%             Last update: 12/02/2019            %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% INITIALIZATION ===================================================================================

clear;
clc;
close all

CurrentFolder = pwd;
idcs          = strfind(CurrentFolder,filesep);
UpperFolder   = CurrentFolder(1:idcs(end)-1);



%% IMAGE LOADING ===================================================================================

% Left images
D_C0 = [UpperFolder '\C0'];
S    = dir(fullfile(D_C0,'*.png')); % pattern to match filenames.

% Right images
D_C1 = [UpperFolder '\C1'];



%% SPECIFY CALIBRATION IMAGES ======================================================================

leftImages = imageSet(D_C0);
imageFileNamesLeft  = leftImages.ImageLocation;
celldisp(imageFileNamesLeft);

rightImages = imageSet(D_C1);
imageFileNamesRight = rightImages.ImageLocation;
celldisp(imageFileNamesRight);

figure(1)
subplot(2,1,1)
montage(imageFileNamesLeft, 'Size',[1 length(S)]);
title('Images taken from C_0 Camera')

subplot(2,1,2)
montage(imageFileNamesRight,'Size',[1 length(S)]);
title('Images taken from C_1 Camera')



%% DETECT THE CHECKERBOARDS ========================================================================
[imagePoints, boardSize] = detectCheckerboardPoints(rightImages.ImageLocation,...
                                                     leftImages.ImageLocation);
```

```
%% PLOT DETECTED POINTS ==================================================================

% Left images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    imshow(imageFileNamesLeft{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    hold on
    plot(imagePoints(:, 1, ii, 2), imagePoints(:, 2, ii, 2), 'go');
    hold on
end

% Right images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    imshow(imageFileNamesRight{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    hold on
    plot(imagePoints(:, 1, ii, 1), imagePoints(:, 2, ii, 1), 'go');
    hold off
end



%% MANUALLY REWRITING OF THE WRONG VALUES ================================================
%{
imagePoints(:,:,2,1) = [];

imagePoints(:,:,1,2) = [];

figure();

subplot(2,5,2)
hold on
plot(imagePoints(:, 1, 2, 1), imagePoints(:, 2, 2, 1), 'go');

subplot(2,5,6)
hold on
plot(imagePoints(:, 1, 1, 2), imagePoints(:, 2, 1, 2), 'go');

%}
```

```
%% ESTIMATION OF THE PARAMETERS ===========================================================

% Specify world coordinates of checkerboard keypoints.
squareSize = 100; % in millimeters
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the stereo camera system.
params = estimateCameraParameters(imagePoints, worldPoints,'EstimateSkew',...
                                  true,'EstimateTangentialDistortion',true,...
                                  'NumRadialDistortionCoefficients',3);

% Visualize calibration accuracy.
figure;
showReprojectionErrors(params);

% Visualize camera extrinsics.
figure;
showExtrinsics(params);

% Plot detected and reprojected points.
figure;
imshow(imageFileNamesRight{1});
title('Detected VS Reprojected Points')
hold on
plot(imagePoints(:, 1, 1, 1), imagePoints(:, 2, 1, 1), 'go');
plot(params.CameraParameters1.ReprojectedPoints(:, 1, 1, 1),
params.CameraParameters1.ReprojectedPoints(:, 2, 1, 1), 'r+');
legend('Detected Points', 'ReprojectedPoints');
hold off

save C1_C0_Params.mat params



%% END PROGRAM_ ===========================================================================
```

9.1.18 – Calibrazione_C1_C2.m – Con settaggio parametri del modello per calibrare

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            STEREO CAMERA CALIBRATION            %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%        Michele Pecchio Ghiringhelli Rota        %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%             Last update: 12/02/2019             %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALIZATION ====================================================================================

clear;
clc;
close all

CurrentFolder = pwd;
idcs          = strfind(CurrentFolder,filesep);
UpperFolder   = CurrentFolder(1:idcs(end)-1);



%% IMAGE LOADING ====================================================================================

% Left images
D_C1 = [UpperFolder '\C1'];
S    = dir(fullfile(D_C1,'*.png')); % pattern to match filenames.

% Preparing right images
D_C2 = [UpperFolder '\C2'];



%% SPECIFY CALIBRATION IMAGES =======================================================================

leftImages = imageSet(D_C1);
imageFileNamesLeft  = leftImages.ImageLocation;
celldisp(imageFileNamesLeft);

rightImages = imageSet(D_C2);
imageFileNamesRight = rightImages.ImageLocation;
celldisp(imageFileNamesRight);

figure(1)
subplot(2,1,1)
montage(imageFileNamesLeft, 'Size',[1 length(S)]);
title('Images taken from C_1 Camera')

subplot(2,1,2)
montage(imageFileNamesRight,'Size',[1 length(S)]);
title('Images taken from C_2 Camera')


%% DETECT THE CHECKERBOARDS =========================================================================
[imagePoints, boardSize] = detectCheckerboardPoints(leftImages.ImageLocation,...
                                                    rightImages.ImageLocation);
```

```
%% PLOT DETECTED POINTS ==================================================================

% Left images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    imshow(imageFileNamesLeft{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesLeft),ii)
    hold on
    plot(imagePoints(:, 1, ii, 1), imagePoints(:, 2, ii, 1), 'go');
    hold on
end

% Right images
for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    imshow(imageFileNamesRight{ii});
    title(['Img ' num2str(ii)]);
end

for ii = 1 : length(imageFileNamesLeft)
    figure(2);
    subplot(2,length(imageFileNamesRight),ii+length(imageFileNamesLeft))
    hold on
    plot(imagePoints(:, 1, ii, 2), imagePoints(:, 2, ii, 2), 'go');
    hold off
end



%% MANUALLY REWRITING OF THE WRONG VALUES ================================================
%{
imagePoints(:,:,2,1) = [];

imagePoints(:,:,1,2) = [];

figure();

subplot(2,5,2)
hold on
plot(imagePoints(:, 1, 2, 1), imagePoints(:, 2, 2, 1), 'go');

subplot(2,5,6)
hold on
plot(imagePoints(:, 1, 1, 2), imagePoints(:, 2, 1, 2), 'go');

%}
```

```
%% ESTIMATION OF THE PARAMETERS ===========================================================

% Specify world coordinates of checkerboard keypoints.
squareSize = 100; % in millimeters
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the stereo camera system.
params = estimateCameraParameters(imagePoints, worldPoints,'EstimateSkew',...
                                   true,'EstimateTangentialDistortion',true,...
                                  'NumRadialDistortionCoefficients',3);

% Visualize calibration accuracy.
figure;
showReprojectionErrors(params);

% Visualize camera extrinsics.
figure;
showExtrinsics(params);

% Plot detected and reprojected points.
figure;
imshow(imageFileNamesLeft{1});
title('Detected VS Reprojected Points')
hold on
plot(imagePoints(:, 1, 1, 1), imagePoints(:, 2, 1, 1), 'go');
plot(params.CameraParameters1.ReprojectedPoints(:, 1, 1, 1),
params.CameraParameters1.ReprojectedPoints(:, 2, 1, 1), 'r+');
legend('Detected Points', 'ReprojectedPoints');
hold off

save C1_C2_Params.mat params



%% END PROGRAM_ ===========================================================================
```

9.1.19 - Script_finaleNoEyesDetection.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%         IMAGE REPROJECTION ON USER CAM          %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%       Michele Pecchio Ghiringhelli Rota         %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%            Last update: 08/02/2019              %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                                                 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALIZATION ===================================================================================

clear
clc
close all

addpath(genpath('Function folder'));
addpath(genpath('Calibration'));

paramsC1_C2 = load('C1_C2_Params.mat'); % Stereo parameters of C1 and C2
paramsC1_C0 = load('C1_C0_Params.mat'); % Stereo parameters of C1 and C0

camList  = webcamlist
nCameras = length(camList);

C1 = webcam(3);                    % WARNING!!
C0 = webcam(1);                    % WARNING!!
C2 = webcam(4);                    % WARNING!!
% preview(C0)
% return



%% DATA & SETTINGS ==================================================================================

% Time counter
disp('%%%% TIME FROM IMAGES LOADING TO REPROJECTION %%%%')

% Video data
nShots = 200;                               % Taken photos number
Resoultn = '320x240';                       % Resolution of each image
C0.Resolution = Resoultn;                   % Resolution of C0 camera
C1.Resolution = Resoultn;                   % Resolution of C1 camera
C2.Resolution = Resoultn;                   % Resolution of C2 camera

% Cloud limits
X_min = -20  ;                              % Min value of detectable X distance, m
X_max =  20  ;                              % Max value of detectable X distance, m
Y_min = -20  ;                              % Min value of detectable Y distance, m
Y_max =  20  ;                              % Max value of detectable Y distance, m
Z_min =  0.05;                              % Min value of detectable Z distance, m
Z_max =  30  ;                              % Max value of detectable Z distance, m

% Disparity settings
BlockSize    = 13;                          % Size of the matching windows,
                                            % (pixels)
BlockSizeVec = [7 13 17];                   % Size of the matching windows (for following test),
                                            % (pixels)
DisptyRnge   = [0 128];                     % [minDisparity maxDisparity],
                                            % (pixel)
UniqssTHold  = 8;                           % Increasing to marking more pixels as unreliable. 15
                                            % (default)
TextrThhold  = 2e-8;                        % Minimum texture threshold. Lower values cause less
                                            % reliable disptyMap. 2e-4 (default)
```

```
% Reduce the elapsed time by increasing the following parameters. Quality may decrease
ptsStep  = 4;                               % Step of the 3D points obtained
sctrStep = 1;                               % Step of the scatter-reconstructed figure.

% Others data
scatterSize = 45;                           % Proportional to the area of the pixel on
mOverpixel  = 6e-6;                         % Length per pixels,              (m/pixel)
CamSize     = .05;                          % Size of the Cameras on the plot, (m)
Hor         = .37;                          % To horizontally shift the maching figure
Ver         = 1e-4;                         % To vertically   shift the maching figure

% Camera C1 Data:
dOC1  = [0 0 0];                            % Coordinates of the camera C1,             (m)
alpha = 0;                                  % Rotation angle around the original X axis, (°)

% Camera C0 Data:
f0 = 2.23/(1+.1/(17*mOverpixel));           % Camera C0 focal length,                   (m)



%% PRELIMINARY CALCULATIONS ===============================================================

% Homogeneous Transformation, betwheen C1 camera and O origin
RxMatrix = rotx(alpha);
RxMATRIX = [RxMatrix [0 0 0]'; [0 0 0 1]];
HomogOC1 = RxMATRIX;
HomogC1O = inv(HomogOC1);

% Homogeneous Transformation, betwheen C1 camera and C0 camera
R_C1_C0 = [paramsC1_C0.params.RotationOfCamera2' [0 0 0]'; [0 0 0 1]];  % Units: (rad)
T_C0_C1 = GenTrMatrix(paramsC1_C0.params.TranslationOfCamera2/1000);    % Units: (m)
HomogC1C0 = R_C1_C0/T_C0_C1;                                            % Equivalent to:
                                                                        % R_C1_C0*inv(T_C0_C1),
                                                                        % which is R_C1_C0*T_C1_C0
HomogC0C1 = inv(HomogC1C0);                                             % Units: (m) and (rad)
dOC0 = HomogOC1\[-paramsC1_C0.params.TranslationOfCamera2/1000 1]';     % Equivalent to: HomogC1O*
                                                                        % [-paramsC1_C0.pa...

% Homogeneous Transformation, betwheen C1 camera and C2 camera
R_C1_C2 = [paramsC1_C2.params.RotationOfCamera2' [0 0 0]'; [0 0 0 1]];  % Units: (rad)
T_C2_C1 = GenTrMatrix(paramsC1_C2.params.TranslationOfCamera2/1000);    % Units: (m)
HomogC1C2 = R_C1_C2/T_C2_C1;                                            % Equivalent to:
                                                                        % R_C1_C2*inv(T_C2_C1),
                                                                        % which is R_C1_C2*T_C1_C2
HomogC2C1 = inv(HomogC1C2);                                             % Units: (m) and (rad)
dOC2 = HomogOC1\[-paramsC1_C2.params.TranslationOfCamera2/1000 1]';     % Equivalent to: HomogC1O*
                                                                        % [-paramsC1_C2.pa...

% Homogeneous Transformation, betwheen C0 camera and origin O
HomogC0O =  R_C1_C0*GenTrMatrix(paramsC1_C0.params.TranslationOfCamera2/1000);

% Understanding Cameras Location and Orientation
figure(1); title('3D Setup'); grid on; hold on; view(-40,30)
xlabel('X,  (m)'); ylabel('Y,  (m)'); zlabel('Z,  (m)');

Cam1 = plotCamera('Location',[0 0
0],'Orientation',HomogOC1(1:3,1:3),'Opacity',0.15,'AxesVisible',true,...
                      'Color','b','Size',CamSize,'Label','C_1');

Cam0 = plotCamera('Location',dOC0(1:3),'Orientation',HomogC1C0(1:3,1:3)*HomogOC1(1:3,1:3), ...
                  'Opacity',0.15,'AxesVisible',true,...
                  'Color','r','Size',CamSize,'Label','C_0');

Cam2 = plotCamera('Location',dOC2(1:3),'Orientation',HomogC1C2(1:3,1:3)*HomogOC1(1:3,1:3), ...
                  'Opacity',0.15,'AxesVisible',true,...
                  'Color','g','Size',CamSize,'Label','C_2');
```

```
%% REAL TIME IMAGE SYNTHESIZING ===================================================================

% Figure preallocating
% figure('units','normalized','outerposition',[0 0 1 1])

for idx = 1 : nShots

    % IMAGE REALLYING ============================================================================

    % Images loading
    tic
    IL = snapshot(C1);
    IR = snapshot(C2);
    t1 = toc;
    fprintf('Images loading.---------- Elapsed time: %.3f sec\n',t1);

%     figure(2)
%     subplot(1,2,1); imshow(IL)
%     title('Captured scene by C_1')
%     subplot(1,2,2); imshow(IR)
%     title('Captured scene by C_2')

    stereoParams = paramsC1_C2.params;
    tic

    [JL, JR] = rectifyStereoImages(IL,IR,stereoParams);%,'OutputView','Valid');
    t2 = toc;
    fprintf('Images rectification.----- Elapsed time: %.3f sec\n',t2);

%     figure(3)
%     subplot(1,2,1); imshow(JL)
%     title('Realigned image from C_1')
%     subplot(1,2,2); imshow(JR)
%     title('Realigned image from C_2')



    % GETTING THE DISPARITY MAP ==================================================================

    % Compute the map
    tic
    JL_bw = rgb2gray(JL);
    JR_bw = rgb2gray(JR);
    disptyMap  = disparity(JL_bw,JR_bw,'BlockSize',BlockSize,'DisparityRange',DisptyRnge);
                          %,'UniquenessThreshold',UniqssTHold);
    t3 = toc;
    fprintf('Disparity computation.---- Elapsed time: %.3f sec\n',t3);
%     Nout = length(disptyMap(:) <= -3.402e+38);    % It varies along the BlockSize

%     figure(4);
%     subplot(1,2,1)
%     imshow(disptyMap, [0, 256]);
%     warning on verbose
%     title(['Disparity Map. BlockSize = ' num2str(BlockSize)]);
%     colormap jet; colorbar
```

```
    % INVERSE PERSPECTIVE GEOMETRY. NEED AT LEAST TWO CAMERAS. NOW WE FIND THE 3D WORLD POINTS OF
    % "F" FROM C1 & C2 POINTS ===========================================================================
    tic
    xyzPoints = reconstructScene(disptyMap,stereoParams);
    xyzPoints = xyzPoints./1000;     % Convert to meter
    t4 = toc;
    fprintf('Getting world points.----- Elapsed time: %.3f sec\n',t4);

    % Create a pointCloud object
    ptCloud = pointCloud(xyzPoints, 'Color', JL);

    % Create a streaming point cloud viewer
%     player3D = pcplayer([-2.5, 2.5], [-4, 1], [0.5, 8], 'VerticalAxis', 'y', ...
%                         'VerticalAxisDir', 'down');
    % Visualize the point cloud
%     view(player3D, ptCloud);



    % SYNTHESIZING OF THE IMAGE PLANE C0 IMAGE ===========================================================

    [t5,t6,points] = FromMapToMatrix(ptCloud, ptsStep, 0);
    fprintf('Reducing points density.-- Elapsed time: %.3f sec\n',t5);
    fprintf('From map form to nx6.----- Elapsed time: %.3f sec\n',t6);

    % Points coordinates whit respect to the C0 camera system
    tic
    Cam_coordsX = HomogC0O(1,1)*points(:,1) + HomogC0O(1,2)*points(:,2) + ...
                  HomogC0O(1,3)*points(:,3) + HomogC0O(1,4)*ones(size(points(:,1)));
    Cam_coordsY = HomogC0O(2,1)*points(:,1) + HomogC0O(2,2)*points(:,2) + ...
                  HomogC0O(2,3)*points(:,3) + HomogC0O(2,4)*ones(size(points(:,1)));
    Cam_coordsZ = HomogC0O(3,1)*points(:,1) + HomogC0O(3,2)*points(:,2) + ...
                  HomogC0O(3,3)*points(:,3) + HomogC0O(3,4)*ones(size(points(:,1)));
    tPart = toc;

    % Reoder points from the closest to the farthest, in order to cover the hidden image points
    tic
    [Cam_coordsX, Cam_coordsY, Cam_coordsZ, points] = SortByDist(Cam_coordsX, Cam_coordsY,...
                                                                 Cam_coordsZ, points);
    t7 = toc;
    fprintf('Sorting pts by distance.-- Elapsed time: %.3f sec\n',t7);

    % Outlayers removing. It can be also useful to remove object over or less than a certain
    % treshold
    tic
    outLay = Cam_coordsX(:) < X_min | Cam_coordsX(:) > X_max;
    Cam_coordsX(outLay) = NaN;
    outLay = Cam_coordsY(:) < Y_min | Cam_coordsY(:) > Y_max;
    Cam_coordsY(outLay) = NaN;
    outLay = Cam_coordsZ(:) < Z_min | Cam_coordsZ(:) > Z_max;
    Cam_coordsZ(outLay) = NaN;
    t8 = toc;
    fprintf('Filtering outlayers.------ Elapsed time: %.3f sec\n',t8);
```

```
    % Coordinates of projected points with respect to image planes C0. Pinhole camera model
    tic
    Img_coordsX = f0*Cam_coordsX./(f0-Cam_coordsZ);
    Img_coordsY = f0*Cam_coordsY./(f0-Cam_coordsZ);
    Img_coordsZ = f0*Cam_coordsZ./(f0-Cam_coordsZ);
    t9 = toc;
    fprintf('Perspective transform.---- Elapsed time: %.3f sec\n',tPart+t9);

    figure(5)
    subplot(1,2,1)
    image(snapshot(C0))
    title('Acquired image from C_0')
    drawnow

    subplot(1,2,2)
    scatter(Img_coordsX(1:sctrStep:end),Img_coordsY(1:sctrStep:end),scatterSize, ...
            points(1:sctrStep:end,4:6),'fill','s');
    fprintf('Time to pixels on.-------- Elapsed time: -ND-  sec\n')
%     axis([[-1 1.1]*1e-3 + 0*Hor , [-8 6]*1e-4 + 1*Ver]);
    view(180,-90)
    axis off
    title('Synthesized image on C_0');
    drawnow

    fprintf('-----------------------------------------------------\n')
    fprintf('TOTAL TIME NEEDED -------- Elapsed time: %.3f sec\n',t1+t2+t3+t4+t5+t6+t7+t8+tPart+t9);
    fprintf('\n\n')

end


%% END_PROGRAM ============================================================================================
```

9.2 – Datasheet fotocamere

9.2.1 – Webcam ELP

DATASHEET DELLE WEBCAM ELP	
ELEMENTO	DETTAGLIO
Model	ELP-USB30W02M-PL37
Sensor	OV7725
Lens Size	1/4 inch
Pixel Size	6.0um X 6.0um
image area	3984 µm x 2952 µm
Max. Resolution	640(H)X480(V)
Compression format	MJPEG / YUV2（YUYV）
Resolution & frame	640X480 MJPEG@ 60fps YUY2@ 30fps/ 480X480 MJPEG@ 30fps YUY2@ 30fps 450X450 MJPEG@30fps YUY2@ 30fps/ 352X288 MJPEG@ 30fps YUY2@ 30fps 320X240 MJPEG@30fps YUY2@ 30fps/ 240X240 MJPEG@ 30fps YUY2@ 30fps
S/N Ratio	50dB
Dynamic Range	60dB
Sensitivity	3.8V/lux-sec@550nm
Mini illumination	0. 2lux
Shutter Type	Electronic rolling shutter / Frame exposure
Connecting Port type	USB2.0 High Speed
Free Drive Protocol	USB Video Class（UVC）
AEC	Support
AEB	Support
AGC	Support
Adjustable parameters	Brightness, Contrast, Saturation, Hue, Sharpness, Gamma, White balance, Backlight Contrast, Exposure
Lens Parameter	3.7mm pinhole lens
Night vision	Need to equipped IR Sensor Lens and IR LED Board
LED board power connector	Support 2P-2.0mm socket
Power supply	USB BUS POWER 4P-2.0mm socket
Power supply	DC5V
Operating Voltage	100mA~160mA
Working current	-10～70℃
Working temperature	-20～85℃
size /Weight	
Cable	Standard 1M / optional 2M,3M,5M
Operating system request	WinXP/Vista/Win7/Win8 Linux with UVC（above linux-2.6.26） MAC-OS X 10.4.8 or later Wince with UVC Android 4.0 or above with UVC

Tabella 14 – Datasheet delle webcam ELP

9.2.2 – Webcam Microsoft

DATASHEET DELLE WEBCAM MICROSOFT	
Version Information	
Product Name	Microsoft® LifeCam VX-2000
Product Version	Microsoft LifeCam VX-2000
Webcam Version	Microsoft LifeCam VX-2000
Product Dimensions	
Webcam Length	2.50 inches (63.5 millimeters)
Webcam Width	1.81 inches (46.0 millimeters)
Webcam Depth/Height	0.92 inches (23.3 millimeters)
Webcam Weight	2.98 ounces (84.5 grams)
Webcam Cable Length	72.0 inches (1829 millimeters)
Compatibility and Localization	
Interface	High-speed USB compatible with the USB 2.0 specification
Operating Systems	Microsoft Windows® 7, Windows Vista®, and Windows XP with Service Pack 2 (excluding Windows XP 64-bit)
Top-line System Requirements	Requires a PC that meets the requirements for and has installed one of these operating systems: • Intel Pentium® 4 3 GHz (Dual Core 1.8 GHz recommended) • 1 GB of RAM (2 GB of RAM recommended) • 1.5 GB hard drive space • Windows-compatible speakers or headphones • USB 1.1 (USB 2.0 recommended). USB port 2.0 required for 1.3 MP and above video capture resolution. You must accept License Terms for software download. Please download the latest available software version for your OS/Hardware combination. Internet access may be required for certain features. Local and/or long-distance telephone toll charges may apply. Software download required for full functionality of all features. Internet functions (post to Windows Live™ Spaces, send in e-mail, video calls), also require: Internet Explorer® 6/7 browser software required for installation; 25 MB hard drive space typically required (users can maintain other default Web browsers after installation)
Compatibility Logos	• Compatible with Microsoft Windows 7 • Certified for Microsoft Windows Vista • Optimized for Microsoft Windows Live • Certified High-Speed USB logo
Software Localization	Microsoft LifeCam software version 3.0 may be installed in Simplified Chinese, Traditional Chinese, English, French, German, Italian, Japanese, Korean, Brazilian Portuguese, Iberian Portuguese, or Spanish. If available, standard setup will install the software in the default OS language. Otherwise, the English language version will be installed.
Windows Live™ Integration Features	
Video Conversation Feature	Windows Live Call button delivers one touch access to video conversation. Photo Swap allows you to share and swap photos during a live video call.
Call Button Life	10,000 actuations
Webcam Controls & Effects	LifeCam Dashboard provides access to animated video special effect features and webcam controls
Blogging Feature	Add photos to Windows Live Spaces with one mouse click
Imaging Features	
Sensor	VGA CMOS sensor technology
Resolution	• Motion Video: 640X480 pixel resolution* • Still Image: 1.3 megapixel interpolated*
Field of View	55° diagonal field of view
Imaging Features	• Digital pan, digital tilt, vertical tilt, and swivel pan, and 4x digital zoom** • Fixed focus from 0.4 m to 2.0 m • Automatic image adjustment with manual override
Product Feature Performance	
Audio Features	Integrated microphone
Mounting Features	Desktop/laptop display mounting hardware
Storage Temperature & Humidity	-40 °F (-40 °C) to 140 °F (60 °C) at <5% to 65% relative humidity (non-condensing)
Operating Temperature & Humidity	32° F (0° C) to 120 °F (40 °C) at <5% to 80% relative humidity (non-condensing)
Certification Information	
Country of Manufacture	People's Republic of China (PRC)
ISO 9001 Qualified Manufacturer	Yes
ISO 14001 Qualified Manufacturer	Yes
Restriction on Hazardous Substances	This device complies with all applicable worldwide regulations and restrictions including, but not limited to: EU directive 2002/95/EC on the Restriction of the Use of Certain Hazardous Substances in Electrical and Electronic Equipment and EU Registration Evaluation and Authorization of Chemicals (REACH) regulation regarding Substances of Very High Concern.
FCC ID	This device complies with Part 15 of the FCC Rules and Industry Canada ICES-003. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation. Tested to comply with FCC standards. For home and office use. Model number: 1381, LifeCam VX-2000.
Agency and Regulatory Marks	• ACMA Declaration of Conformity (Australia and New Zealand) • ICES-003 report on file (Canada) • EIP Pollution Control Mark, EPUP (China) • CE Declaration of Conformity, Safety and EMC (European Union) • WEEE (European Union) • VCCI Certificate (Japan) • CITC Letter (Kingdom of Saudi Arabia) • KCC Certificate (Korea) • GOST Certificate (Russia) • UkrSEPRO Certificate (Ukraine) • FCC Declaration of Conformity (USA) • UL and cUL Listed Accessory (USA and Canada) • CB Scheme Certificate (International)
Windows Hardware Quality Labs (WHQL)	ID: 1436289 Microsoft Windows 7

Tabella 344 – Datasheet delle webcam Microsoft

10 - RICONOSCIMENTI

Il presente documento è stato redatto in circa sei mesi di studi, consulenze, e messe a punto. Ritengo importante sottolineare che è soltanto osservando *attivamente* quello che ci circonda che si può riuscire a migliorarne qualcosa. Nel mio caso, l'idea che ho voluto accingermi a sviluppare, rinominata "*Sistema di identificazione scena stradale*", è scaturita a seguito di un minimo di esperienza alla guida. Diverse volte, mi è infatti capitato di riuscire ad intravedere motoveicoli o biciclette (situate negli angoli ciechi della mia visuale) soltanto a seguito di spostamenti della mia testa o più in generale di spostamenti relativi tra i miei occhi e tali veicoli. Mi sono così reso conto di una necessità, e come ingegnere ho iniziato a pensare come poterla soddisfare.
In generale, non ritengo servano anni di università per poter avere idee ammirevoli, reputo piuttosto che sia sufficiente avere una buona volontà nel contribuire alla crescita in generale. D'altra parte, invece, l'inevitabile parte di sviluppo ed i mezzi per tentare di arrivare al fine, spesso impongono conoscienze tipiche si svariati rami dell'ingegneria. In questo caso, in virtù della natura sperimentale della tesi ma soprattutto dato il campo applicativo abbastanza differente dal mio percorso di studi, sono state molteplici le alternanze tra periodi di fallimento e di successo. Per questa ragione, sento di dover almeno ringraziare tutti coloro che sono stati sempre pronti ad ascoltarmi o incoraggiarmi.

Non si vuole ora applicare una scala di merito tra tutte le persone che mi hanno sostenuto nel corso di questi ultimi anni, ma sicuramente meritano quantomeno di essere citati:

- **Berganti Andrea** --- : Per l'interesse mostrato sulla mia tesi, per essere un buon amico
- **Calderoni Paolo** --- : Per la spontaneità con cui dai aiuto e vuoi far star bene le persone
- **Carlappi Andrea** --- : Per l'amicizia dimostrata, per i consigli che hai dato nell'ambito ingegneristico e nella vita
- **Dacrema Matteo** -- : Per la prontezza con cui hai risposto ai dubbi che ho avuto in ambito informatico, perchè senza esitare mi hai regaleto 3 webcam Microsoft
- **Gaiaschi Marco** ---- : Per l'interesse dimostrato nei miei confronti, per la volontà di ascoltarmi ed essere ascoltato; perchè mi hai prestato il Samsung
- **Inzadi Erica** --------- : Per essere stata sempre pronta ad ascoltare le scoperte che ho ritenuto più interessanti, per aver gioito nel contribuire alla mia tesi: hai realizzato schemi, immagini; per tutto quello che sei
- **Saturno Riccardo** -- : Per l'immensa simpatia, pazienza e amicizia mostrata da sempre. Ad ogni incontro degli ultimi mesi, hai voluto gli aggiornamenti della tesi

Ringrazio tutti gli amici del gruppo "**La Mandria**" e del gruppo "**Quelli del Lazzaretto**".

Ringrazio i professori ***Mauro Stefano*** e ***Sorli Massimo*** per avermi ascoltato, appoggiato, consigliato e gratificato durante lo sviluppo della tesi.

Ringrazio infine, ovviamente, **la mia Famiglia**, alla quale devo gran parte di quello che sono.