Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA ED ARCHITETTURA

Corso di Laurea in Ingegneria dell'automazione - Curriculum: Automation Engineering- 0931

STRUMENTI DI GUIDA OTTICA PER UN SISTEMA DI MARCATURA LASER

Tesi di Laurea Magistrale in Ingegneria dell'Automazione

Candidato:
FABRIZIO PETTINARI

Relatore:
Chiar.mo Prof. LUIGI DI STEFANO

Correlatore:
Chiar.mo Ing. GRAZIANO IMBRIACO

Sessione II

Anno Accademico 2016/2017

"Al mio Gianca,
ad Andrea e
... non ultima alla mia mamma."

Introduzione

Il percorso di tesi che ho intrapreso è stato svolto presso l'azienda Datalogic ©, con l'intento di integrare un sistema di visione ad un sistema di marcatura laser.
Nell'ambiente industriale dei giorni d'oggi, l'uso di un marcatore laser è sempre più diffuso. Oggetti di qualunque dimensione ed impiego vengono marcati per scopi che vanno dal semplice datamatrix, alla marcatura di loghi più complessi. L'utilizzo di questo potente strumento è però vincolato dalla particolare posizione fisica occupata, di volta in volta, dall'oggetto; per questo motivo viene fissato nella posizione desiderata, attraverso dime meccaniche.
Fin ad ora si riteneva assolutamente necessaria la presenza di un operatore per il controllo del corretto posizionamento, tramite una simulazione della marcatura. Per ovviare a questo limite strutturale, Datalogic ha pensato di introdurre uno strumento di aiuto e di visione del processo: la camera. L'idea di base è stata quella di impiegare le moderne smart camera per individuare l'oggetto da marcare e rendere quindi il processo più automatico possibile. Per giungere a questo risultato è stato necessario effettuare una calibrazione del sistema totale: Camera più Laser. Il mio studio si è focalizzato quindi nel creare un eseguibile che aiutasse il cliente ad effettuare questa operazione nella maniera più semplice possibile. E' stato creato un eseguibile in C# che mettesse in comunicazione i due dispositivi ed eseguisse la calibrazione dei parametri intrinseci ed estrinseci. Il risultato finale ha permesso di avere il sistema di riferimento mondo della camera coincidente con quello del piano di marcatura del laser. Ne segue che al termine del processo di calibrazione se un oggetto viene rilevato dalla camera, avente il baricentro nella posizione (10,10), il laser, utilizzando le medesime coordinate, marcherà proprio nel baricentro dell'oggetto desiderato. La maggiore difficoltà riscontrata è stata la differenza

dei software che permettono la comunicazione con i due dispositivi e la creazione di una comunicazione con il laser, non esistente prima in C#.
Nel primo capitolo si descrivono gli strumenti ed i software utilizzati. Nel capitolo due viene descritta il tipo di comunicazione creata. Infine, verrà analizzato il percorso che ci ha portato alla creazione dell'eseguibile, ed i vantaggi operativi ottenuti.
Nell'immagine riportata è possibile vedere l'intero sistema, formato dal laser UniQ e dalla camera posta lateralmente, leggermente inclinata, in modo che sia visibile l'intera area di marcatura del laser.

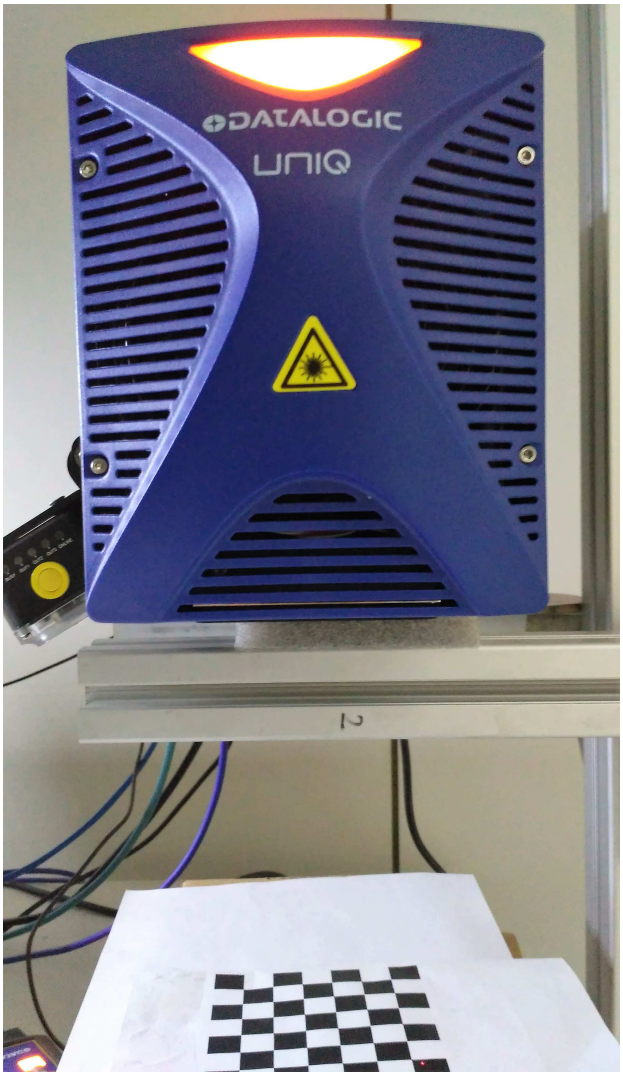


Figura 1: Sistema

Abstract

My thesis has been carried out with Datalogic ©. The final aim of the project is to integrate a laser into a camera system. A P-series, ultra-compact, smart camera has been used as it offers advanced machine vision functionalities and is a fully embedded stand-alone device. The other device, UniQ, is an ultra-compact laser marking system based on fiber laser technology. Until now, in order to mark an object, operators had to put it manually in the marking area of the laser and had to check the correct position through simulation. The company would benefit with the addition of an intelligent device, such as the smart camera, as it would eliminate the need of a manual presence of an operator. The computer vision tool is able to perfectly detect the object and mark it. Challenges associated with the project have been: the types of programs used by each device, and their different reference frames. We used the language C# to establish the communication with the devices and to calibrate the whole system. We started our application using methods of calibration used in Impact. However, due to the novelty of this project, we had to write and develop a script using C# in order to establish communication with the laser. At the end, we have created a unique, executable file, that is able to calibrate the system automatically by estimating the intrinsic and extrinsic parameters.

Indice

Introduzione **i**

1 Strumenti utilizzati **1**

1.1 Smart Camera 1

1.2 Laser Marker 4

1.3 Funzionamento di un Camera 8

1.4 Strumenti di calibrazione disponibili 15

2 Sistema Sviluppato **19**

2.1 Comunicazione Camera 20

2.2 Comunicazione Laser 22

2.2.1 Comandi Laser 26

3 Sviluppo di un nuovo strumento di calibrazione **29**

3.1 Algoritmi Precedenti 29

3.2 Analisi della Target Calibration 31

3.3 Analisi della Real World Coordinates 32

3.4 Applicazione Sviluppata 36

4 Risultati e Conclusioni **43**

Riferimenti bibliografici 46

Bibliografia **47**

Elenco delle figure

1 Sistema ii

1.1 P-Series 2
1.2 Programmi Laser 3
1.3 UniQ 4
1.4 Funzionamento laser 5
1.5 Programmi Laser 7
1.6 Proiezione Prospettica Pinhole Camera 9
1.7 Utilizzo Lente 10
1.8 Digitalizzazione Immagine 11
1.9 Immagine Rototraslazione 12
1.10 Calibrazione metodo di Zhang 14
1.11 Target Calibration 16
1.12 Calibrazione WRC 17

2.1 "Porta" dell'SDK 20
2.2 JavaScript 22
2.3 Comandi da inviare al Laser 25
2.4 Comando Generico 25
2.5 Comando di Risposta 26
2.6 Comando per Aprire Documento dal Device 27
2.7 Comando per Muovere e Ruotare Documento 27
2.8 Comando per Muovere e Ruotare un Oggetto 28
2.9 Riassunto Movimento Oggetto 28

3.1 Step Calibrazione 30
3.2 Pattern 31
3.3 Scacchiera Modificata con punti 33
3.4 Scacchiera Modificata 34
3.5 Tempi di Apertura Diaframma 35
3.6 Interfaccia eseguibile 36

4.1 Blob Detection 45

Elenco delle tabelle

1.1 Perfomance del Laser 6
1.2 Qualità della calibrazione 18

4.1 Perfomance del metodo ‘‘FixShutterAndFindPoint()’’ 44

Capitolo 1

Strumenti utilizzati

In questo primo capitolo vengono illustrati gli strumenti operativi che sono stati utilizzati per questo progetto di tesi. Infatti il sistema finale raggiunto vede l'utilizzo di diversi software e l'interazione di dispositivi dalle caratteristiche differenti. In primo luogo verrà analizzata la camera ed i software che permettono di gestirla, dopodiché l'utilizzo del laser ed infine verrà illustrato in breve, il funzionamento di una camera e la sua calibrazione.

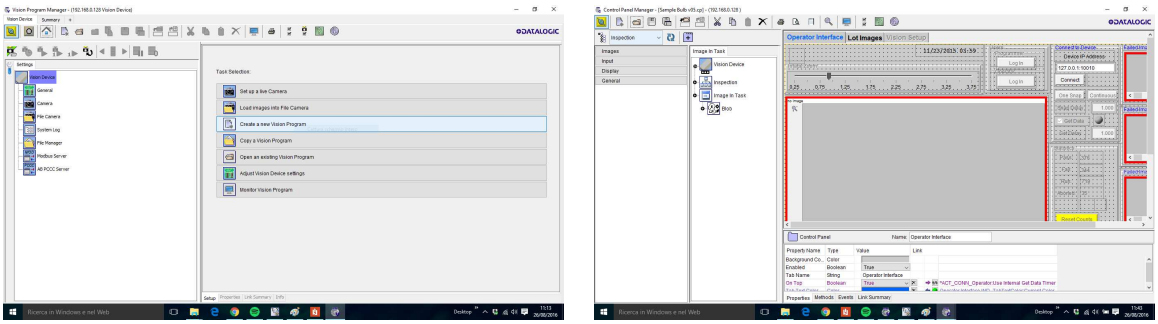
1.1 Smart Camera

La Smart camera utilizzata nel progetto è la P-Series [1]. Questo prodotto offre avanzate funzioni di visione artificiale in un dispositivo stand-alone completamente integrato. Sono presenti diverse versioni di P-Series: 10, 12, 15, 17. Per il nostro scopo é stata utilizzata la versione 10 con sensori di immagine CMOS in scala di grigi con risoluzione VGA (640 x 480) e avente dimensione del pixel pari a $5,3\,\mu$m. E' stata adottata questa soluzione poiché rappresenta il caso peggiore su cui poter testare un processo di calibrazione. La minore risoluzione della camera diminuisce il numero di informazioni utilizzabili per l'algoritmo di calibrazione, il ché porta ad un facile utilizzo dell'algoritmo sviluppato anche su camere con migliori caratteristiche. Sono presenti, infatti, anche altre versioni con immagini a colori e risoluzione di 1.3 MP(1280 x 1024), classificate come P-Series 15/17. Per far fronte ad ambienti di scarsa e non omogenea luminosità,

Figura 1.1: P-Series

il dispositivo supporta anche una serie di illuminatori intercambiabili e sostituibili direttamente dall'utente. Come é visibile nella figura 1.1, la camera alloggia in una cover protettiva di alluminio dalle dimensioni di 95 x 54 x 43mm e pesante circa 238g; valutata secondo lo standard IEC 60529 con un codice IP (International Protection) pari a IP67, cioè totalmente protetto contro la polvere, sabbia e anche da immersioni temporanee. Permette inoltre un utilizzo operativo fino alla temperatura massima di 50°C [2]. Queste particolari caratteristiche la rendono adatta ad un ambiente industriale, e le sue possibili applicazioni vanno dall'industria automobilistica, elettronica, farmaceutica ed alimentare. E' alimentata tramite una CBX (Connection Box); scatola di connessione industriale progettata per rendere più facili le operazioni di cablaggio dei dispositivi Datalogic e che permette sia di fornire l'alimentazione ma anche di elaborare trigger derivanti da sensori esterni. Poiché il dispositivo in questione non supporta la funzione di autofocus, é necessaria la messa a fuoco manuale attraverso la regolazione di una vite e l'utilizzo "Live View" del programma VPM (Vision Program Manager). Quest'ultimo é uno dei tre programmi utilizzabili per interagire con la camera. Difatti, tramite la presenza del connettore Ethernet, é possibile collegare un pc per poter gestire le potenzialità della camera attraverso tre programmi: Vision Program Manager, in figura 1.2a, Control Panel Manager in figura 1.2b, e Impact SDK (Software Development Kit).

- **VPM:** è il programma principale per poter configurare la camera. Permette, prima



(a) VPM (b) CPM

Figura 1.2: Programmi Laser

di tutto, di settare i parametri principali (apertura shutter, trigger, illuminatori, calibrazione...) per un corretto funzionamento a seconda delle condizioni operative dell'utente. In secondo luogo, è necessario per poter creare il task di ispezione desiderato. Creando un file di ispezione si avrà la possibilità di utilizzare i diversi tool presenti. I tool vanno dal semplice Blob detection al più complicato Pin Point Tool. Combinando tutti questi tool l'utente potrà individuare in modo univoco un particolare oggetto che verrà acquisito dalla camera e volendo, fornire le coordinate di esso ad un altro dispositivo. Il file di ispezione che si viene a creare viene chiamato vision program (.vp).

- **CPM:** questo programma invece serve per creare l'interfaccia utente in maniera da essere utilizzata in real time mentre all'interno della camera il file di ispezione, precedentemente configurato, esegue le sue operazioni. Permette quindi, ad esempio, di vedere a video i risultati dell'ispezione e di modificare alcune configurazioni in maniera rapida. Infatti rispetto ai complessi linguaggi di programmazione, CPM offre la massima flessibilità e consente di creare pannelli di controllo in tempi decisamente inferiori a quelli normalmente richiesti. Rappresenta la vera e propria HMI (Human Machine Interface) della camera.

- **IMPACT SDK:** quest'ultimo non è un vero e proprio software. E' una libreria scritta in C# che integrata ad un progetto di Visual Studio permette di utilizzare delle classi e metodi per interagire con la camera. Attraverso queste APIs (Application Program Interface) l'utente può sviluppare la proprio HMI perfettamente

adattata al suo scopo. Quindi il CPM e la libreria SDK vengono utilizzate per lo stesso fine anche se quest'ultima richiede una maggiore abilità di programmazione (conoscenza del C #) ma permette una maggiore flessibilità. Tramite i metodi è possibile infatti poter richiamare qualunque tool della camera già utilizzato nel VPM. Le principali APIs di cui si può usufruire sono:

- Connessione/disconnessione ad un Vision Device.
- Caricare programmi di ispezione.
- Caricare o scaricare immagini dal/verso il pc.
- Ricavare dati dall'immagine.
- Calibrare la camera.
- Scattare fotografie.

1.2 Laser Marker

Figura 1.3: UniQ

Viene di sotto riportato il funzionamento del Laser Marker utilizzato, e dei due software necessari per la marcatura. Il prodotto scelto è UniQ^{TM}, in figura 1.3, basato su tecnologia Fiber Laser, che consente di marcare, per marcatura, abrasione o incisione, i dati identificativi di prodotti come codici a matrice 2D, codice a barre lineari, codici postali, codici a barre stacked, testo con qualsiasi formato di font corrispondente agli

standard vettoriali, numeri di matricola alfanumerici, codici particolari, grafica e loghi in qualsiasi ambiente di produzione[3]. Il Laser Marking è un processo che permette di modificare la fisica di un materiale in maniera irreversibile, mediante un intenso raggio laser che viene focalizzato sulla superficie dell'oggetto da marcare. Grazie all'utilizzo di specchi motorizzati, controllati tramite software, figura 1.4, è possibile modificare l'angolo e posizione del raggio, in maniera tale di creare l'immagine voluta sulla superficie del nostro componente . Durante questo processo viene focalizzato un raggio laser aven-

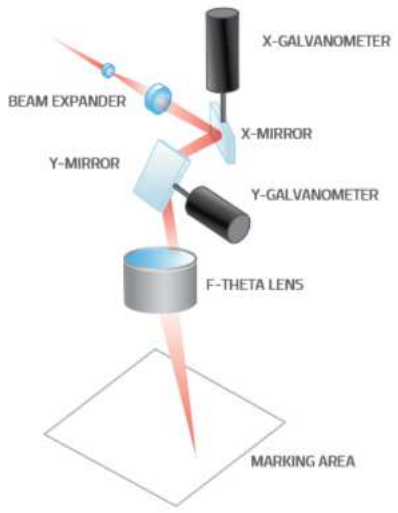


Figura 1.4: Funzionamento laser

te un'istantanea densità di potenza, pari a qualche centinaio di chilowatt che causa la modifica del materiale. Uno tra i notevoli vantaggi di questa soluzione è che il Laser Marker in questione non prevede l'utilizzo di inchiostri, acidi, solventi; il che porta ad un basso impatto ambientale. Anche la perdita di materiale è minimale, poiché viene abrasa solamente la parte superficiale dell'oggetto, non modificando in maniera massiccia lo spessore del componente.

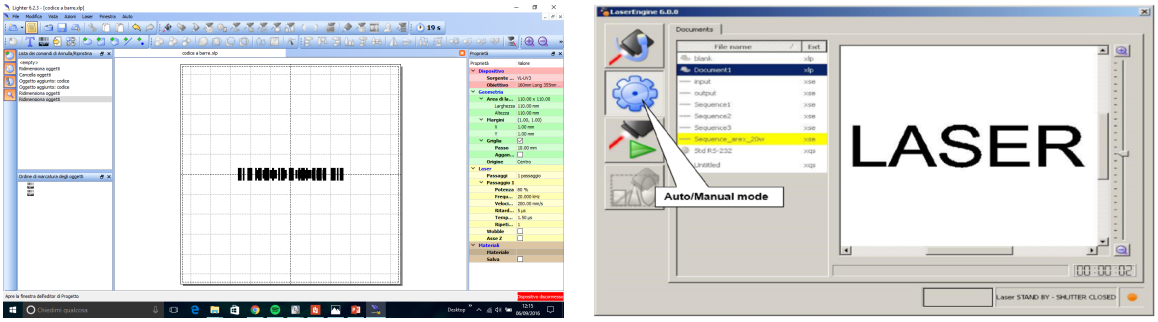
Le dimensioni dell'area di lavoro costituiscono un'impostazione di importanza fondamentale. Esse dipendono da diversi elementi, tra cui il tipo di lente (F-Theta) utilizzata e la relativa distanza focale, con l'aumentare della distanza focale aumenterà parallelamente l'area di lavoro. Nel caso specifico è stato utilizzato un UniQ avente una lente con una distanza focale pari a 160mm, corrispondente ad un area di marcatura di 110x110mm;

anche qui il livello di protezione è elevato: IP54, difesa contro polvere e umidità. Grazie alla elevata potenza di $15W$, i marcatori laser UniQ offrono una grande flessibilità applicativa e permettono di marcare efficacemente un vastissimo campo di materiali. Il design compatto, ed il sistema di raffreddamento assiale, silenzioso ed efficace, permette una rapida e semplice installazione anche nelle situazioni più complesse ed in spazi ristretti. I laser generalmente sono classificati in base al livello di pericolo per la salute umana. Le norme attuali dividono i laser in 7 classi, dipendenti anche dalla potenza e lunghezza d'onda. Questo dispositivo è classificato di livello 4, poiché la radiazione laser accessibile è molto pericolosa per gli occhi e per la pelle. Per questi motivi si deve evitare di esporre occhi o pelle alla radiazione diretta o diffusa, è necessario quindi l'uso di lenti adatte durante il suo utilizzo. Nella tabella 4.1 vengono illustrati alcune caratteristiche principali del nostro dispositivo. Sono presenti anche due diodi laser che emettono un fascio lumi-

Tipo di Laser	Classe 4
Potenza Nominale (W)	15
Energia Di Impulso (max) (mJ)	0,75
Potenza di Picco (max) (kW)	10
Potenza di Rete Richiesta ($Volt$)	100/240
Lunghezza d'onda centrale d'emissione (nm)	1064
Frequenza di Ripetizione (kHz)	15 100
Velocità d Marcatura (mm/sec)	Fino a 5000
Area di Lavoro (mm^2)	100x100
Distanza di Lavoro (mm)	183+-5

Tabella 1.1: `Perfomance del Laser`

noso rosso: quello centrale, sfruttando la serie di specchi e lenti utilizzati per il processo di marcatura, proietta il centro dell'area di lavoro del laser ed effettua la simulazione di ciò che si vuole incidere; l'altro, laterale, è impiegato principalmente per capire quando il piano su cui si vuole marcare risulta a fuoco. Per ottenere una messa a fuoco ottimale, una volta azionati i diodi, viene spostato verticalmente il piano di marcatura fin quando i due punti proiettati andranno a coincidere, raggiunta questa situazione il tutto risulterà



(a) Laser Editor (b) Laser Engine

Figura 1.5: Programmi Laser

perfettamente a fuoco e non si avrà pericolo di rifrazione.
Di notevole interesse è l'assenza di un unità di controllo o alimentazione esterna. Questo dispositivo supporta infatti un pc embedded gestito dal sistema operativo Windows Embedded, il che permette di eliminare l'utilizzo di un processore esterno. Grazie a questa caratteristica è possibile avere due modalità di connessione: Locale e Remota. La prima permette di collegare i dispositivi di output/input (mouse, tastiera e monitor) direttamente al pc embedded; mentre la seconda permette la gestione, utilizzando un pc esterno connesso tramite un cavo di rete. Andando a completare la descrizione dell'utilizzo del laser, vengono descritti i due software necessari: Laser Editor e Laser Engine, rispettivamente in figura 1.5a e 1.5b [4].

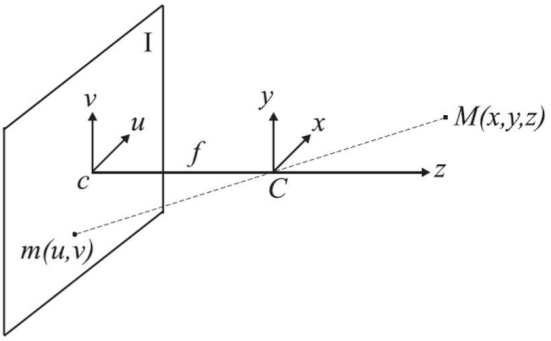
- **Laser Editor:** è il software necessario per la configurazione del file di marcatura. Con questo l'utente può creare l'oggetto di marcatura voluto, partendo dalle più semplici forme geometriche (cerchio, quadrato, linee), arrivando ad elementi più complessi come codici a barre lineari o bidimensionali. Per semplificare il lavoro è anche possibile importare dei loghi o immagini di tipo vettoriali. Al di fuori della qui descritta fase di editing di layout grafico, questo software permette anche di configurare dei parametri importanti da inviare al laser. Può essere scelto il tipo di materiale da marcare, il numero di passaggi della luce laser sull'oggetto da incidere, la percentuale di potenza massima da utilizzare, la frequenza e la velocità di scansione. Consente inoltre la creazione di procedure automatizzate da integrare facilmente in linee di montaggio con o senza PC. Infine da qui è possibile inviare

in marcatura il layout o , per essere sicuri del corretto posizionamento dell'oggetto nell'area di marcatura, visualizzare i limiti, ovvero verificare i margini del disegno nel piano di lavoro tramite l'utilizzo del puntatore laser rosso

- **Laser Engine:** è il software che consente di interagire direttamente con il dispositivo laser. Infatti questo software non può essere installato in un dispositivo esterno, ma solamente nel pc Embedded del laser. L'interfaccia prevede di vedere in anteprima come verrà marcato il documento precedentemente creato con il Laser Editor. Esso ha varie modalità operative, offrendo un controllo del laser sia in modalità remota che in modalità locale.

1.3 Funzionamento di un Camera

Ora viene analizzato nel dettaglio il funzionamento di una camera [5]. La camera rappresenta un particolare dispositivo che permette di catturare i raggi riflessi derivanti da un oggetto tridimensionale. Durante questo processo di cattura viene persa una dimensione dell'oggetto presente nel mondo reale. Questa operazione, chiamata proiezione prospettica, consiste in una vera e propria proiezione dei punti dell'oggetto reale (spazio tridimensionale) su di un piano identificato dal sensore digitale della camera (spazio bidimensionale). Una proiezione prospettica può essere modellata tramite due sistemi differenti. Il primo, più semplice, riprende il modello chiamato Pinhole camera; mentre il secondo tiene conto anche della non idealità del sistema, introducendo il concetto di lente. La pinhole camera è un sistema teorico nel quale è presente un unico foro di dimensioni infinitesimali attraverso cui un raggio luminoso passa e proietta l'immagine tridimensionale. Nella figura 1.6 si può vedere schematicamente come viene idealizzato il processo. Con la lettera **I** viene evidenziato il piano immagine, mentre f indica la distanza tra quest'ultimo ed il piano focale. Il piano focale rappresenta il piano su cui giace il sistema di riferimento mondo, caratterizzato dalla terna di coordinate (x, y, z) avente origine in **C**, centro ottico. Il raggio luminoso attraversa il centro ottico (unico foro presente) e giunge sul piano immagine in maniera da creare il corrispettivo punto immagine. La formula 1.1 descrive il concetto precedentemente esposto: dove x, y, z individuano le coordinate del punto tridimensionale dell'oggetto mentre u e v indicano

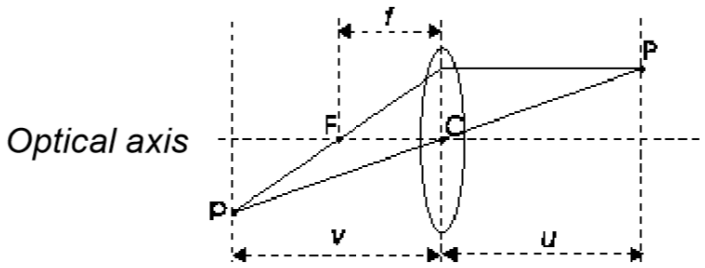


$$\begin{cases} u = -x\frac{f}{z} \\ v = -y\frac{f}{z} \end{cases} \tag{1.1}$$

Figura 1.6: Proiezione Prospettica Pinhole Camera

le coordinate del punto bidimensionale proiettato. Si può notare come la distanza focale gioca un ruolo cruciale per la posizione del punto immagine. Il vantaggio teorico di questo sistema è che la profondità di campo, la zona in cui gli oggetti nell'immagine appaiono ancora nitidi e sufficientemente focalizzati, è infinita. Il sistema qui descritto è considerato solamente a livello ideale. Difatti nella realtà un foro infinitesimale farebbe passare cosi poca luce che per poter proiettare l'oggetto sarebbe necessario un tempo di esposizione lunghissimo, considerando anche il conseguente problema dell'indispensabile staticità della scena.

Per questi motivi i sistemi camera utilizzati prevedono l'utilizzo di una lente che focalizza la luce entrante nel piano immagine, questa permette infatti di raccogliere più luce e consente di avere immagini più nitide anche con un basso tempo di esposizione. Tuttavia la profondità di campo qui non è più infinita, come nel caso precedente del pinpoint camera. Il modello lente è visibile in figura 1.7, dove è possibile notare come il punto mondo **P**, essendo a fuoco, venga proiettato in **p** tramite la nota proiezione prospettica. Nella figura precedente 1.7 è stata indicata con $\boldsymbol{f}$ la focale della lente, mentre l'effettiva lunghezza focale della proiezione prospettica per i punti a fuoco risulta qui indicata come v. Solamente il piano alla distanza u dalla lente è a fuoco. Ne consegue quindi che i punti del mondo reale giacenti in quel piano saranno a fuoco, mentre tutti gli altri saranno sfuocati, creando il cosiddetto cerchio di confusione. Per ovviare a questo limite è presente un meccanismo che permette di avvicinare o allontanare la lente in maniera da modificare la distanza focale. Dopo aver illustrato i concetti base di un camera, vengono



$$\frac{1}{u} + \frac{1}{u} = \frac{1}{f} \tag{1.2}$$

Figura 1.7: Utilizzo Lente

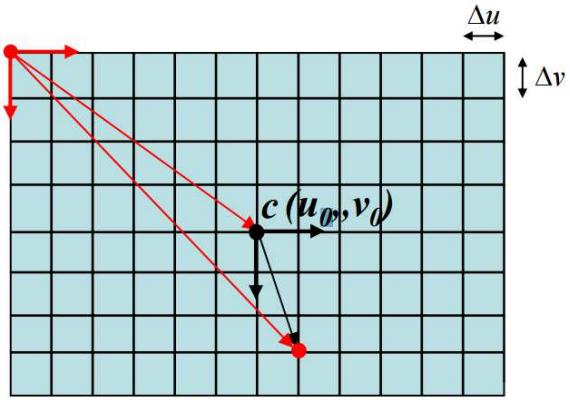
di sotto riportati alcuni passaggi matematici per evidenziare parametri importanti della calibrazione. Per semplificare la precedente formulazione 1.1, è stata effettuata la traslazione del piano immagine al di la del centro ottico, in maniera da ottenere la seguente formulazione con segno positivo. $\begin{cases} u = x\frac{f}{z} \\ v = y\frac{f}{z} \end{cases}$

Un ulteriore step matematico consiste nel passare da un sistema di coordinate euclideo ad un sistema di coordinate omogeneo. Un punto nell'ambiente reale è descritto come un vettore tridimensionale, spazio euclideo $\Re^3$. Aggiungendo una coordinata alla nostra terna (**x y z**) e moltiplicando tutti gli elementi per uno stesso fattore moltiplicativo K, otteniamo però una nuova rappresentazione dello stesso punto nello spazio proiettivo, P^2. Quindi la nuova quadrupla sarà cosi formata: (K**x** K**y** K**z** K) $\forall K \neq 0$. Il passaggio allo spazio proiettivo permette di rappresentare anche punti che vanno all'infinito, rappresentati dal valore 0 nell'ultima coordinata. Questa trasformazione consente inoltre un importante vantaggio, come si può vedere nella seguente formulazione 1.3 espressa in coordinate omogenee; la proiezione prospettica può essere scritta come una trasformazione lineare.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f\frac{x}{y} \\ f\frac{y}{z} \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.3}$$

Passando alla notazione matriciale $k\tilde{\mathbf{m}} = \tilde{\mathbf{P}}\tilde{\mathbf{M}}$, dove i vettori $\tilde{\mathbf{m}}$, $\tilde{\mathbf{M}}$ rappresentano rispettivamente il punto immagine ed il punto reale espressi entrambi in coordinate omogenee,

è possibile vedere come la $\tilde{\mathbf{P}}$ rappresenta il modello geometrico della camera, propriamente detto matrice di proiezione prospettica **(PPM)**. Un altro importante aspetto da analizzare è la digitalizzazione dell'immagine, visibile in figura 1.8, tramite la quale si ottiene un nuovo sistema di riferimento, non più centrato nel punto principale "c", ma traslato in un angolo del sensore. Infatti il piano immagine risulta essere discretizzato in base alla dimensione dei pixel: caratterizzato da Δu e Δv, rispettivamente dimensione orizzontale e verticale del pixel. Per questi motivi il sistema verrà descritto tramite la nuova formula 1.4: dove u_0 ed v_0 rappresentano le coordinate del centro immagine rispetto al nuovo sistema di riferimento. Attraverso questa rappresentazione è possibile



$$\begin{cases} u = \frac{1}{\Delta u}\frac{f}{z}x + u_0 \\ v = \frac{1}{\Delta v}\frac{f}{z}y + v_0 \end{cases} \tag{1.4}$$

Figura 1.8: Digitalizzazione Immagine

giungere ad una matrice, $\mathbf{A}$, che contenga tutte le caratteristiche del sensore. Passando di nuovo alle coordinate omogenee, si ottiene la formula descritta in 1.5, tramite la quale è possibile scomporre la matrice $\tilde{\mathbf{P}}$ per ottenere la matrice dei parametri intrinseci $\mathbf{A}$.

$$\tilde{\mathbf{P}} = \begin{bmatrix} f\frac{1}{\Delta u} & 0 & u_0 & 0 \\ 0 & f\frac{1}{\Delta v} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f\frac{1}{\Delta u} & 0 & u_0 \\ 0 & f\frac{1}{\Delta v} & v_0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & u_0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \mathbf{A}\left[\mathbf{I}\middle|\mathbf{0}\right] \tag{1.5}$$

La matrice $\mathbf{A}$ conterrà quindi tutti i parametri intrinseci: u_0, v_0, $1/\Delta u$, $1/\Delta v$, f e lo "skew", che rappresenta l'angolo tra gli assi del sistema di riferimento del sensore. Un ulteriore aspetto di cui dobbiamo tenere conto è il fatto che in certe situazioni il sistema

di riferimento della finale non coincide con quello della camera, come si può vedere in figura 1.9. Per questo motivo è necessario applicare una rototraslazione (**R**, **T**), descritto

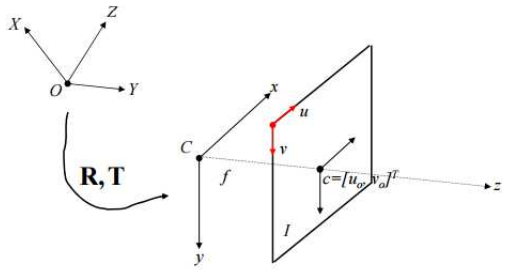


Figura 1.9: Immagine Rototraslazione

nella formula 1.6, dove **W** indica la terna di coordinate del punto mondo espressa rispetto al nuovo sistema di coordinate esterno alla camera(World Reference Frame).

$$\mathbf{W} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \mathbf{M} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \mathbf{M} = \mathbf{RW} + \mathbf{T} \tag{1.6}$$

Passando poi alla rappresentazione omogenea, 1.7, è più facile ottenere la matrice **G** dei parametri estrinseci, composta da sei valori indipendenti: tre per la traslazione e tre per la rotazione. Questi ultimi facenti parte della matrice **R**, corrispondente ai relativi angoli di rotazione rispetto ai 3 assi (**x**, **y**, **z**).

$$\tilde{\mathbf{W}} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tilde{\mathbf{M}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \tilde{\mathbf{M}} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \tilde{\mathbf{W}} = \mathbf{G}\tilde{\mathbf{W}} \tag{1.7}$$

Nell'equazione 1.8 è possibile riassumere la relazione che porta da un sistema di riferimento mondo ad un sistema di riferimento immagine, mettendo in evidenza le due matrici dei parametri.

$$k\tilde{\mathbf{m}} = \mathbf{A}\left[\mathbf{I}\middle|\mathbf{0}\right]\begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}\tilde{\mathbf{W}} \tag{1.8}$$

La modellizzazione del sistema camera fin qui descritta non tiene conto di un fattore molto importante: la presenza della lente. Infatti in un sistema reale la lente introduce

una deviazione considerevole dovuta a due fenomeni: la distorsione radiale e quella tangenziale. La distorsione radiale è causata dalla curvatura della lente, mentre quella tangenziale dal decentramento dei componenti del sistema e dai difetti di produzione. A causa di questi fattori si deve quindi introdurre la definizione di punto immagine reale: influenzato dalla lente ed effettivamente "osservato" nel piano immagine, e la definizione di punto immagine ideale, identificato dalla posizione teorica del punto mondo, proiettato nel caso di una lente non deformante.
La seguente formula, 1.9, mostra come tramite una relazione non lineare i due valori vengono legati. $L(r)$ indica la distorsione radiale mentre i fattori $d\tilde{x}$, $d\tilde{y}$ la distorsione tangenziale.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = L(r) \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} + \begin{bmatrix} d\tilde{x} \\ d\tilde{y} \end{bmatrix} \tag{1.9}$$

Sono stati fin qui introdotti tutti gli aspetti funzionali che saranno necessari a comprendere la calibrazione di una camera. Prima di procedere oltre, vengono ora brevemente riassunti gli step che permettono di passare da un punto mondo ad un punto immagine.

- Considerando un punto 3D, passare dal sistema di riferimento mondo (WRC) al sistema di riferimento della camera (CRF) utilizzando i parametri estrinseci.
- Effettuare la proiezione prospettica in maniera da passare da un punto 3D ad un punto immagine.
- Passare dal punto immagine ideale al punto immagine reale applicando la deformazione della lente.
- Passare dal sistema di riferimento 2D standard a quello espresso in coordinate pixel, applicando la matrice degli intrinseci.

Un algoritmo di calibrazione permette di stimare tutti i parametri sconosciuti (intrinseci ed estrinseci) di una camera. Per effettuare quest'operazione è necessario scattare delle foto a degli oggetti di forma nota; in questa maniera è più facile trovare la corrispondenza tra punto 2D e punto 3D.
Esistono due metodi di calibrazione che utilizzano dei pattern di forma nota: il primo

prevede l'impiego di un unica immagine contenente molti piani; mentre il secondo l'impiego di un pattern piano fotografato da molte posizioni differenti. Si illustra di seguito l'algoritmo di calibrazione di Zhang, il più classico di cui normalmente ci si avvale, con l'impiego di un pattern classico: la scacchiera[6]. Per applicare questo metodo devono essere noti: il numero interno delle righe e colonne della scacchiera e la lunghezza del lato del quadrato che la compone. Ad ogni frame viene fissato come sistema di riferimento l'angolo superiore sinistro della scacchiera, vedi figura 1.10, avente il piano z=0 coincidente con il piano di giacimento del pattern. Con le assunzioni precedentemen-

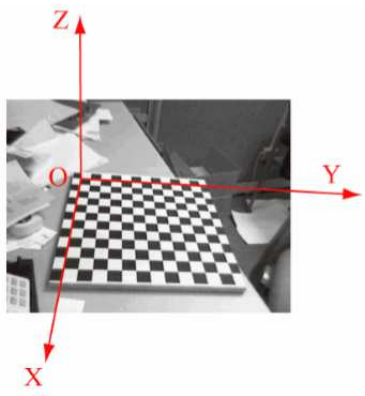


Figura 1.10: Calibrazione metodo di Zhang

te descritte, si lavora solamente sui punti noti della scacchiera, considerando quelli 3D aventi tutti la coordinata z=0. Questa considerazione porta ad un notevole vantaggio, diminuendo la complessità della trasformazione, poiché in questo modo la proiezione prospettica diventa un omografia. Cioè, si raggiunge una semplice trasformazione lineare tra piani. Per poter trovare tutti i nove parametri costituenti questa omografia è necessario sfruttare la conoscenza delle coordinate degli m corner della scacchiera ed i corrispondenti punti immagine rilevati tramite un apposito algoritmo di detection, quale Harris corner detector. Avremo quindi a disposizione tre equazioni (solamente due di queste linearmente indipendenti) con nove incognite per ogni corner della scacchiera. Il sistema che si viene a creare è quindi formato da $2m$ equazioni in 9 incognite. Tramite questa operazione si raggiunge una prima stima dell'omografia, ottenuta minimizzando un errore algebrico.
Utilizzando questa stima iniziale si può migliorare la valutazione dell'omografia risolven-

do ai minimi quadrati il problema non lineare basato sull'errore di tipo geometrico. Per cui si cerca di minimizzare l'errore presente tra il punto immagine realmente rilevato e il corrispondente punto immagine stimato, utilizzando l'omografia iniziale.
Si sottolinea che i passaggi sopra elencati si riferiscono ad una singola immagine; poiché, se si considerasse un differente punto di vista, cambierebbe di conseguenza anche l'omografia. A partire da queste stime e sfruttando il fatto che i parametri intrinseci sono invece gli stessi per tutte le immagini, é possibile arrivare alla stima della matrice **A**. Si genera quindi un sistema lineare di $2n$ equazioni in sei incognite, con n pari al numero di immagini. Determinati i parametri intrinseci, si ricaveranno quelli estrinseci per ogni differente frame.
Da cui, si calcolerà la stima la stima dei coefficienti di distorsione, avvalendosi della conoscenza delle coordinate reali (distorte) dei corner, ricavate direttamente dall'immagine, e delle coordinate ideali (non distorte) provenienti dalla proiezione dei punti 3D mediante l'omografia stimata.

1.4 Strumenti di calibrazione disponibili

In questa sezione illustreremo gli strumenti di calibrazione fornitaci da Impact e gestibili tramite il VPM. In particolare, utilizzando questi strumenti, è possibile ricavare la distorsione prospettica, il "pixel size" e la rototraslazione applicata per giungere al nuovo sistema di riferimento da noi imposto. Il "pixel size" invece indica il fattore di scala che è applicato alla nostra immagine, cioè a quanto corrisponde un pixel nella realtà. Sono presenti due tipi di calibrazione possibili: la "Target Calibration" e la "Real World Coordinates" che andremo a descrivere nel dettaglio.

- **Target Calibration:** questo strumento permette di identificare la distorsione prospettica, della lente e ricavare il fattore di scala. Come è possibile vedere in figura 1.11 sono utilizzabili tre diversi pattern (Hexagon, Checkerboard, Grid) da dover stampare su semplice carta. L'utente deve selezionare quale pattern andrà ad utilizzare e per poter identificare anche il "pixel size", inserire la dimensione reale dell'oggetto calibrato ("pitch"), differente in base al pattern scelto. Ad esempio, per quanto riguarda il pattern scacchiera, questo valore di' "pitch" corrisponde al-

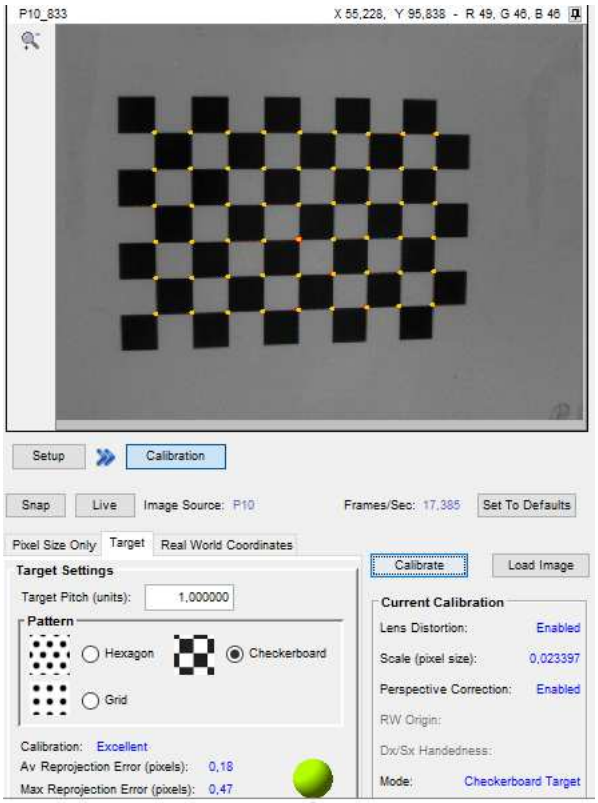


Figura 1.11: Target Calibration

la dimensione reale del lato di un quadrato della scacchiera espressa in millimetri. Dopodiché si potrà lanciare la calibrazione ponendo, in maniera centrata, il target stampato al di sotto della camera. Cosi facendo viene eseguito uno scatto, che attraverso un apposito procedimento algoritmico, va a ricavare i parametri intrinseci. In questo caso la distanza focale f è un parametro noto della lente utilizzata. Per ottenere una buona calibrazione il pattern deve essere ben visibile, per questo motivo devono essere settati due parametri fondamentali: il tempo di apertura dell'otturatore e gli illuminatori. Regolando, in maniera opportuna, questi due parametri è possibile rende l'immagine più nitida possibile. In particolare gli illuminatori sono consigliati per ottenere più contrasto possibile tra il pattern e lo sfondo bianco.

- **Real World Coordinates:** tramite quest'altro strumento è invece possibile far

Figura 1.12: Calibrazione WRC

si che il sistema di riferimento camera venga rototraslato in un sistema di riferimento da noi imposto. In prima battuta è necessario rilevare dei punti notevoli (in numero maggiore di quattro) tramite lo strumento: "Generates Point". Questo metodo permette di identificare i baricentri dei blob presenti nell'immagine scattata dalla camera. Dopo aver determinato tutti i baricentri, rispetto al sistema di riferimento camera precedente, l'utente dovrà inserire le nuove coordinate dei baricentri rispetto al nuovo sistema di riferimento voluto. Cosi facendo l'algoritmo avrà, per ogni baricentro, sia le coordinate del vecchio che del nuovo sistema di riferimento. Quindi è molto importate che questo binomio di coordinate sia calcolato in maniera precisa. Nella figura 1.12 viene mostrata l'interfaccia di calibrazione in questione. In questo esempio sono presenti cinque cerchi di cui vengono rilevati i baricentri; l'utente andando selezionare ognuno di essi dovrà inserire le nuove

coordinate che andranno a comporre la parte della tabella indicata da **RW X** e **RW Y**. Come vedremo nell'ultimo capitolo, questa procedura è stata la parte più difficile da automatizzare, poiché l'algoritmo di "Generates Point" identifica tutti i baricentri degli oggetti presenti nella scena ma effettuando la ricerca con ordine non sempre uguale. Quindi se nella scena sono presenti tre oggetti non è detto che la prima coppia di coordinate ricavate dall'algoritmo sia associata sempre al primo di questi.

Qualità della calibrazione	Average Reprojection Error Value
Eccellente	$0 \div 0.5$
Ottima	$0.5 \div 1.5$
Discreta	$1.5 \div 3.0$
Bassa	$3.0 \div 5.0$

Tabella 1.2: `Qualità della calibrazione`

Questi due metodi, illustrati fin ora, restituiscono un valore su cui poter comprendere la validità e la buona riuscita dell'algoritmo: **Average Reprojection Error** e **Maximum value Reprojection Error**. L'errore di riproiezione indica quanto si discosta il punto immagine rilevato dal punto immagine stimato. Il punto immagine stimato è calcolato applicando la proiezione prospettica utilizzando i parametri precedentemente stimati. Quindi il primo valore rappresenta la media degli errori di riproiezione; mentre il secondo valore indica il massimo errore di riproiezione tra tutti i punti della scena. Questi danno una buona stima di quanto siano esatti i parametri determinati, quanto più piccoli sono i valori di questi due fattori tanto più precisa è la stima. Nella tabella 1.2 viene illustrato il range dei valori assunti dall"'average reprojection error" e la relativa qualità della stima.

Queste due interfacce di Impact qui illustrate vengono riprese poi nell'ultimo capitolo dove si è voluto sfruttare la potenzialità di ciò che già esisteva per creare un sistema automatizzato utilizzando il C#.

Capitolo 2

Sistema Sviluppato

Dopo aver descritto nel dettaglio i dispositivi utilizzati, in questo capitolo andremo ad illustrare il tipo di sistema sviluppato per poter raggiungere il nostro scopo; cioè quello di creare un unico eseguibile che lanciandolo permetta di giungere alla calibrazione del sistema camera più laser. In particolare si vedrà in quale maniera é stata gestita la comunicazione con essi. La scelta del metodo di comunicazione è stata sicuramente influenzata dal possibile utilizzo dei programmi già preesistenti. La camera può essere gestita, come specificato nel primo capitolo, tramite diversi software; mentre il laser, oltre ai già citati programmi, può interpretare dei comandi esadecimali ricevuti, ed agire opportunamente. Il sistema iniziale da cui si è partiti prevede l'utilizzo di un pc esterno, tramite il quale si sono testati i vari comandi da inviare ad entrambi i dispositivi. A causa della minore versatilità della camera si è scelto di utilizzare la libreria SDK "VisionSDK", basata sul linguaggio di programmazione C#, che ha permesso di creare, oltre ad un ambiente fortemente personalizzato, adatto alle nostre esigenze, un eseguibile con interfaccia grafica in Windows Form. Questa scelta però ci ha portati a dover progettare la parte di comunicazione laser, non essendo questa mai stata affrontata precedentemente con un linguaggio di programmazione come il C#. Per questo motivo, sono state realizzate due principali classi: la prima, denominata "Camera", racchiude solo metodi ed oggetti presi dall'sdk; l'altra, "Laser", è stata invece scritta completamente da zero. Infine, per poter collegare il pc ai due dispositivi si è utilizzato uno switch. L'introduzione di quest'ulteriore strumento si è resa necessaria a causa dell'unica porta di rete presente

nel pc. Per ovviare a questo limite, successivamente, si è tentato di eliminare l'impiego del computer esterno, facendo funzionare l'eseguibile direttamente sul pc Embedded del laser, con esito positivo. Ne risulta quindi, che nel processo di calibrazione non sarà più indispensabile la presenza di un calcolatore esterno.

2.1 Comunicazione Camera

Per prima cosa la camera é stata collegata tramite un cavo Ethernet al nostro switch. E' possibile settare l'indirizzo IP della camera tramite VPM, altrimenti di default la camera é in ascolto all'indirizzo "192.168.0.128". Come precedentemente esposto, per la comunicazione con la camera, si è scelta la libreria "VisionSDK", basata sul framework 3.5 di Microsoft .NET, sfruttando il protocollo di comunicazione binario Thrift. Come già accennato in precedenza, esistevano delle classi e metodi, appartenenti al namespace "VisionSDK", che permettevano di replicare qualunque funzione già presente nel VPM di Impact. Questa libreria,"VisionSDK", è progettata in maniera da poter richiamare facilmente i vari tool presenti nel VPM di Impact. Infatti per l'utilizzo di una qualunque funzione presente nel VPM sarà sufficiente usare la corrispettiva "porta". La cosiddetta "porta" è rappresentata da una stringa che può essere ricavata dalla proprietà del comando VPM utilizzato.

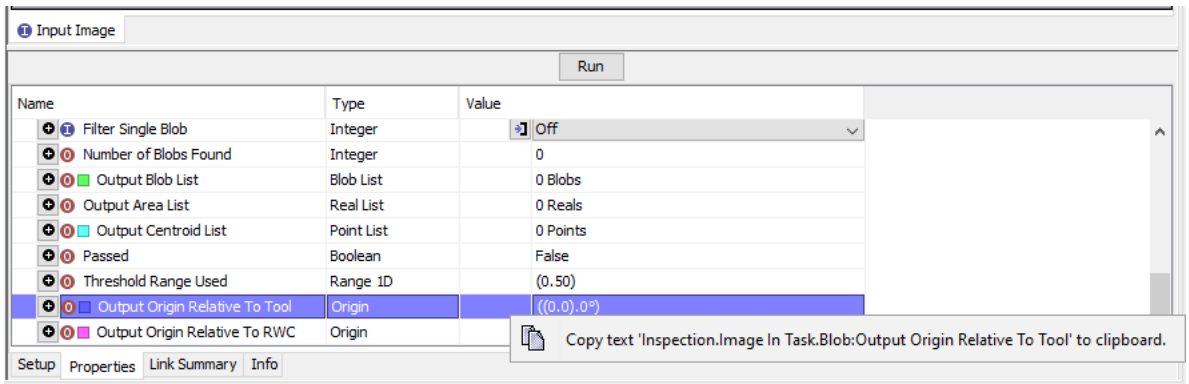


Figura 2.1: "Porta" dell'SDK

Come si vede in figura 2.1 per tutti i comandi è presente il tipo di dato restituito e la relativa stringa, la "porta" da utilizzare. Nell'esempio specifico, nel caso di utilizzo del

metodo “Output Origin Relative To Tool” che restituisce la posizione del baricentro di un blob, si può notare che il dato è di tipo Origin e la relativa stringa da copiare per utilizzare quel comando è: ”Inspection.Image In Task.Blob:Output Origin Relative To Tool”. Grazie a questo meccanismo, analizzando bene Impact, è facile replicare la stessa funzione con l’sdk.
Procedendo nell’analisi del namespace “VisionSDK” si può vedere che sono presenti diverse classi; le principali utilizzate sono: “VisionDevice” ed “Operation Result”. La prima classe permette di essere associata al dispositivo Camera utilizzato ed interagire con esso. Tra le molteplici funzionalità presenti, si sono sfruttati maggiormente i seguenti metodi: la connessione alla camera, il settaggio del tempo di esposizione, l’abilitazione/-disabilitazione degli illuminatori, le funzioni di calibrazione, ed il “Generates Point”. La classe di “Operation Result” permette, invece, di gestire i risultati delle funzioni utilizzate nella classe precedente. Infatti ogni metodo della classe “VisionDevice” restituisce un risultato in output che notifica il successo o insuccesso dell’operazione svolta. Questo risultato può essere gestito in maniera tale da rendere noto all’utente la buona/cattiva riuscita dell’operazione, o può essere processato direttamente dal programma così da prendere delle decisioni di conseguenza.
I metodi della classe “VisionDevice” possono essere sia di tipo asincrono che sincrono. Quelli asincroni permettono di utilizzare una particolare funzione in maniera tale da non bloccare il processore fino al conseguimento di essa. Per l’algoritmo utilizzato si è optato però per l’utilizzo di metodi sincroni data la particolare sequenzialità del processo di calibrazione. Questi ultimi aspettano la restituzione del risultato, dato dall’oggetto “Operation Result”, prima di procedere con l’esecuzione dell’istruzione successiva. In questa maniera ogni metodo attende il conseguimento dell’operazione associata prima di procedere oltre. Tornando al lavoro svolto, la classe creata, “Camera”, permette di racchiudere una lista di funzioni della classe “VisionDevice” sotto un unico metodo di questa nuova classe. Questo, così ottenuto, conterrà diverse istruzioni necessarie per ottenere un unico scopo. In questa maniera il codice risulta ben strutturato e più compatto. Ad esempio, per effettuare la calibrazione dei parametri intrinseci si devono utilizzare alcuni metodi della classe “VisionDevice”: si setta la dimensione della scacchiera, il tipo di pattern utilizzato e si scatta la foto. Questo elenco di operazioni viene inglobato nell’unico

metodo "targetCalibrationSync(int pitch)", illustrato in seguito nel capitolo 3.2, della classe "Camera".

2.2 Comunicazione Laser

In questa sezione si illustra la parte di comunicazione Laser, effettuata tramite l'utilizzo di un PC esterno, mai svolta con linguaggio di programmazione C#. Prima di procedere con la spiegazione tecnica della comunicazione effettuata, bisogna precisare che si è dovuto decidere a priori quale dei due dispositivi effettuasse l'operazione di Server, e quale di Client. In particolare, il server è colui che attende una richiesta da servire, mentre il client effettua tale richiesta. Per questo motivo, il programma principale scritto in C# e supportato dal pc esterno, rappresenta il client che invia i messaggi al programma Server, scritto in Java e presente nel pc embedded del laser, per poter effettuare delle particolari operazioni. Lato server si è dovuto creare un JavaScript che aprisse la porta di comunicazione, "2709", e restasse in ascolto di eventuali messaggi inviati da dispositivi connessi alla rete, figura 2.2.

Figura 2.2: JavaScript

Mentre lato client si è creata la classe "Laser" che a breve verrà illustrata.
Per poter creare questa interazione PC esterno e Laser si è scelto di utilizzare il protocollo di comunicazione, "TCP/IP" (Transmission Control Protocol/Internet Protocol) [7]. Un protocollo di comunicazione è definito, in generale, come una raccolta di regole che vengono redatte in maniera tale da stabilire una comunicazione corretta tra due programmi distinti . I diversi protocolli di comunicazione sono organizzati secondo un sistema detto a "Livelli": Fisico, Datalink, Rete, Trasporto, Sessione , Presentazione, Applicazione [8]. Il protocollo "TCP/IP" comprende due livelli di questo stack: trasporto e rete. Il TCP è classificato a livello di trasporto e garantisce che tutti i pacchetti inviati ad un computer remoto siano realmente arrivati a destinazione. Invece l'IP è valutato a livello di rete e stabilisce le regole base per instradare i pacchetti di dati che viaggiano in una rete. Questo protocollo utilizza un indirizzo di rete ed un numero di porta. L'indirizzo individua un dispositivo specifico nella rete; il numero di porta, invece, identifica il particolare servizio del dispositivo a cui fare riferimento. In realtà, si poteva utilizzare anche un altro protocollo di comunicazione: "UDP" (User Datagram Protocol). Quest'ultimo consente una comunicazione molto più rapida ma non garantisce la corretta ricezione dei dati; invece l'altro, "TCP/IP", qualora i pacchetti inviati non siano giunti a destinazione, li rispedisce al mittente fino alla corretta ricezione; motivo per cui abbiamo virato la scelta verso quest'ultima soluzione.
Dopo aver illustrato il tipo di connessione effettuata si è studiato come poterla far funzionare in un ambiente di programmazione .NET. Questo framework mette a disposizione delle classi per poter gestire in maniera più efficiente il protocollo "TCP/IP", In particolare all'interno della classe creata, "Laser", si è utilizzata la classe "Socket" [9]. Essa supporta due modalità di funzionamento: quella sincrona ed asincrona. In modalità sincrona, le chiamate alle funzioni che eseguono operazioni di rete (ad esempio "Send" e "Receive") attendono che quest'ultime si completino, prima di restituire il controllo al programma chiamante. In modalità asincrona, invece, il programma chiamante non viene sospeso ma continua con la propria esecuzione. Solitamente viene utilizzata quest'ultima modalità in programmi processanti immagini, gestendo la connessione di rete su di un thread separato mentre l'applicazione vera e propria continua ad essere in esecuzione sul thread principale. In questo modo nel momento in cui viene inviato un

messaggio il nostro processore può continuare ad effettuare delle altre operazioni, non rimanendo bloccato in ascolto del messaggio di risposta. Questo è un notevole vantaggio poiché l'elaborazione di un immagine può impiegare diverso tempo durante il quale non è consigliato tenere in stallo il processore, soprattutto se si vuole garantire il real time dell'applicazione. Questa classe, "Socket", necessità di essere inizializzata, tramite l'utilizzo del costruttore, con le informazioni sull'indirizzo di rete e sul protocollo utilizzato (nel nostro caso TCP/IP). La combinazione di indirizzo di rete e porta viene chiamata: endpoint; rappresentata in .NET Framework dalla classe "EndPoint". Della classe principale "Socket" sono stati utilizzati i metodi "BeginReceive" e "BeginSend", che permettono rispettivamente, in maniera del tutto asincrona, di iniziare l'invio del messaggio e di notificare la ricezione.
Prima di questo progetto era però presente una lista di comandi esadecimali da poter inviare al laser in maniera tale da eseguire delle particolari operazioni. Infatti, attraverso il Javascript in esecuzione nel pc Embedded, è possibile interpretare correttamente questi comandi in maniera tale che ogni stringa esadecimale corrisponde ad una precisa operazione del laser. In figura 2.3 sono elencati tutti i comandi preesistenti che permettono di interagire con il laser; in giallo sono stati evidenziati i comandi che sono stati trascritti in C#, trattati nel dettaglio nella sottosezione 2.2.1 [10]. Il tipo di comando da inviare è rappresentato da una stringa esadecimale che viene convertito poi in un vettore di byte per poter essere inviato al laser. Sono presenti due tipi di messaggi. I messaggi con formato prefissato (marcare, fermare la marcatura...) e quelli con lunghezza e contenuto variabile. Per i secondi la procedura di creazione del messaggio automatico è stata più complicata a causa della variabilità dei parametri da inserire all'interno di esso. Ogni parametro inserito deve essere convertito seguendo la tavola Ascii, cioè assegnando al parametro l'associato valore esadecimale della tavola. Questo comporta che la lunghezza del messaggio sarà direttamente proporzionale al numero di cifre decimali o dal numero di lettere del parametro inserito. Ad esempio nel momento in cui devo inserire come parametro un nome di un file uguale a "prova.xlp", questo deve essere convertito per ottenere il corrispettivo valore esadecimale. In questo caso otterrò la sequenza pari a: "70 72 6F 76 61". La generalizzazione del comando è rappresentata in figura 2.4, dove si può vedere che ogni comando è caratterizzato da un byte di "escape", da un identificatore

General Commands	
0xF1 81	Get Version
0xF1 82	Get Version (verbose)
0xF1 91	Get laser status
0xF1 92	Get laser status (verbose)
0xF1 93	Get protocol error
0xF1 94	Get protocol error (verbose)
0xF1 A1	Get system time
0xF1 A2	Set system time
Files and I/O Handling	
0xF2 81	Get document list
0xF2 82	Open document from device
0xF2 83	Open document from file system
0xF2 84	Save document
0xF2 91	Set I/O port
0xF2 92	Get I/O port
Data Handling	
0xF3 81	Get global counter list
0xF3 82	Get global string list
0xF3 83	Get global counter value
0xF3 84	Set global counter value
0xF3 85	Get global string value
0xF3 86	Set global string value
0xF3 91	Enable/disable data field
0xF3 92	Set data field value
0xF3 93	Get data field value
0xF3 96	Set imported field value
0xF3 A2	Move and rotate document
0xF3 A1	Move data field
0xF3 A6	Move and rotate data field
0xF3 98	Get Objects IDs
0xF3 A4	Get document parameters
0xF3 A5	Set document parameters
Axis Handling	
0xF5 81	Move axis
0xF5 82	Reset axis
0xF5 83	Is axis in home position
0xF5 84	Get axis range
0xF5 85	Get axis position
0xF5 86	Is axis enabled
0xF5 87	Stop axis
0xF5 88	Check axis movement
Laser Handling	
0xF5 E1	Start laser test
0xF5 E2	Stop laser test
0xF5 F1	Start aiming
0xF5 F2	Start marking
0xF5 FF	Stop marking

Figura 2.3: Comandi da inviare al Laser

0x1B	<msg length>		<command class>	<command>	<parameters>	<msg terminator>	
	Low Byte	High Byte	1 byte	1 byte	0...N byte	0x0D	0x0A

Figura 2.4: Comando Generico

di comando, da dei parametri e da un fine messaggio. La procedura standard da effettuare prevede di inviare un comando e di aspettare il corrispettivo comando di risposta inviato dal laser. Questa procedura risulta utile utilizzando una successione di comandi, infatti possiamo terminare la comunicazione nel caso in cui non ci sia nessuna risposta da parte del laser. Come si può vedere in figura 2.5, il comando di risposta del Laser è composto dall'acknowledgement che serve ad indicare se i dati ricevuti sono coerenti o meno con quanto inviato. Per comprendere quando il messaggio ricevuto è terminato, si aspetta fino alla ricezione della sequenza dei due byte di "escape": "0D 0A". Infatti nel momento in cui viene rilevata questa sequenza si esce dal metodo "ReceiveCallback" che

permetteva di rimanere in ascolto alla porta indicata. Nella risposta è presente anche il tipo di errore se la procedura non è andata a buon fine. Per analizzare il messaggio di risposta è stato creato un metodo apposito nella classe "Laser": "CheckErrorCode", il quale serve ad analizzare il secondo byte della stringa, cioè il Low Byte indicato in figura 2.5. Se questo è uguale a "08" significa che ho avuto un errore e controllando anche la parte di "error code" posso capire di che genere è stato, altrimenti se è uguale a "04" il messaggio è andato a buon fine.

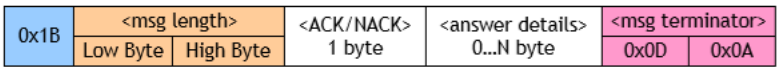


Figura 2.5: Comando di Risposta

2.2.1 Comandi Laser

- **Get Laser Status:** Questo comando viene utilizzato per conoscere lo stato in cui si trova il laser. Il laser infatti potrebbe trovarsi in uno dei seguenti stati: laser off, laser warm up, laser wait for start, laser standby, laser standby shutter closed, laser ready, laser ready shutter closed, laser emission, laser busy shutter closed, laser warning, laser error. Eseguendo questo comando l'utente può capire lo stato del sistema e prendere le successive decisioni in maniera automatica.

- **Open Document From Device:** Permette di caricare un file .xpl, precedentemente creato, archiviato nel seguente path del Laser: "Lighter Data\Docs\Layouts". Con questo comando è possibile quindi richiamare un dato Layouts facendo riferimento al nome del file. Come rappresentato in figura 2.6, il messaggio deve essere strutturato ponendo la stringa, opportunamente convertita in esadecimale, nella posizione occupata da "File Name". Per questo scopo è stato creato il metodo: "public byte[] CreateMessageOpenDocumentFromDevice(String path)" che permette di prendere in ingresso una stringa, di convertirla e di creare il suddetto messaggio da poter inviare successivamente al laser.

- **Move and Rotate Document:** Permette di ruotare e muovere l'intero documento rispetto la sua origine. Come si vede in figura 2.7, il messaggio deve essere

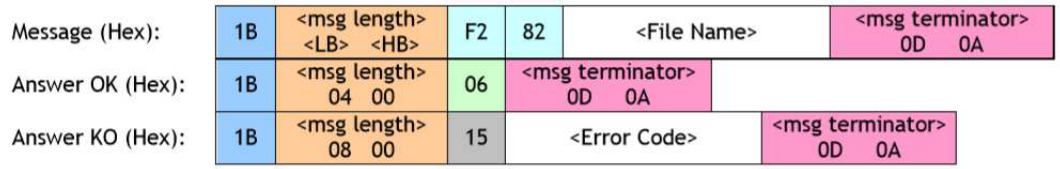


Figura 2.6: Comando per Aprire Documento dal Device

composto dai due offset(X, Y) e dall'angolo. Il relativo metodo di creazione di questo messaggio quindi avrà in ingresso tre valori interi indicanti i tre rispettivi parametri.

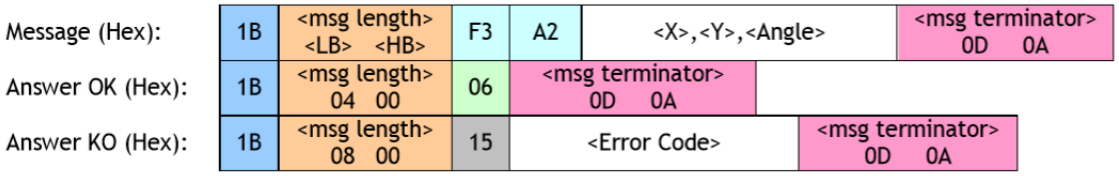


Figura 2.7: Comando per Muovere e Ruotare Documento

- **Move and Rotate Data Field:** Permette di ruotare e muovere un oggetto, presente nel file precedentemente caricato, identificato da un preciso numero. Infatti in un progetto Lighter è possibile associare un particolare numero identificativo a ciascun oggetto presente nel layout. Come si può vedere in figura 2.8, la creazione di questo messaggio è più complicata. Infatti la lunghezza del messaggio non è prefissata poiché dipende dal numero di cifre dei valori inseriti. Si voleva avere un metodo che andasse a posizionare in maniera automatica i relativi campi del messaggio. E' stato quindi creato il metodo: "public byte[] CreateMessageMoveDataFieldPoint(int iD, Point point, Double angle)" sempre della classe Laser, che prendesse in ingresso l'identificativo dell'oggetto da spostare, le nuove coordinate e l'angolo di rotazione. Le coordinate vengono arrotondate ad un numero decimale con tre cifre significative oltre la virgola e poi viene creato il messaggio nella maniera descritta. Nell'immagine, mostrata in figura 2.9, invece è possibile osservare come l'oggetto venga rototraslato rispetto al centro di marcatura del laser; il co-

mando passato per effettuare questa operazione prevede un offset lungo X di 10, lungo Y di 5, ed una rotazione di 45°.

Message (Hex):	1B	<msg length> <LB> <HB>	F3	A6	<Field Name>	0A	<X>	0A	<Y>	0A	<Angle>	<msg terminator> 0D 0A
Answer OK (Hex):	1B	<msg length> 04 00	06	<msg terminator> 0D 0A								
Answer KO (Hex):	1B	<msg length> 08 00	15	<Error Code>								<msg terminator> 0D 0A

Figura 2.8: Comando per Muovere e Ruotare un Oggetto

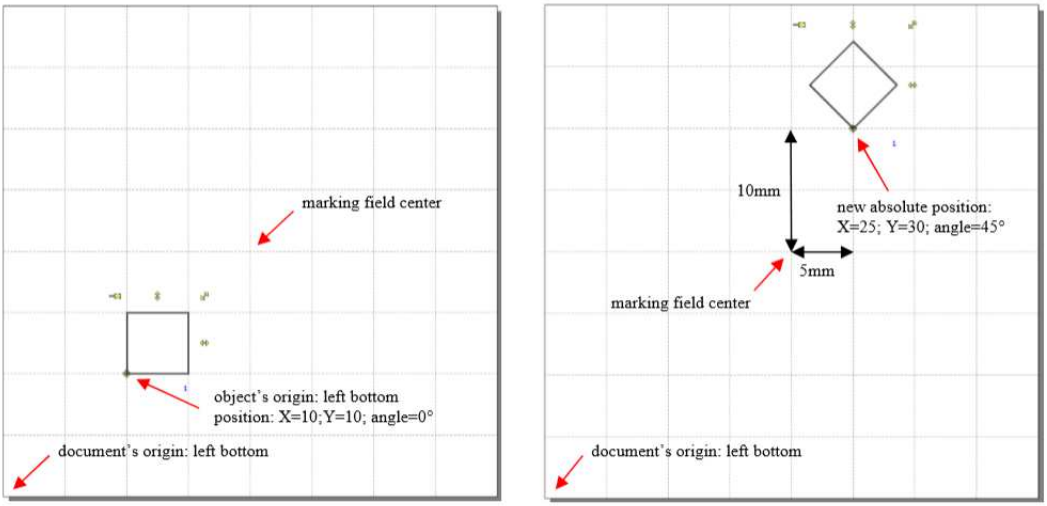


Figura 2.9: Riassunto Movimento Oggetto

- **Start Aiming:** Permette di proiettare i limiti del documento caricato. I limiti rappresentano un rettangolo che racchiude interamente l'oggetto da marcare. Questo metodo viene lungamente utilizzato in fase di montaggio del pezzo da marcare, infatti grazie a questa proiezione è possibile vedere dove il laser andrà a marcare, ed eventualmente correggere la posizione dell'oggetto.
- **Start Marking:** Permette di iniziare il processo di marcatura.
- **Stop Marking:** Permette di interrompere il processo di marcatura.

Da ricordare che tutti i valori di offset precedentemente elencati sono espressi in millimetri.

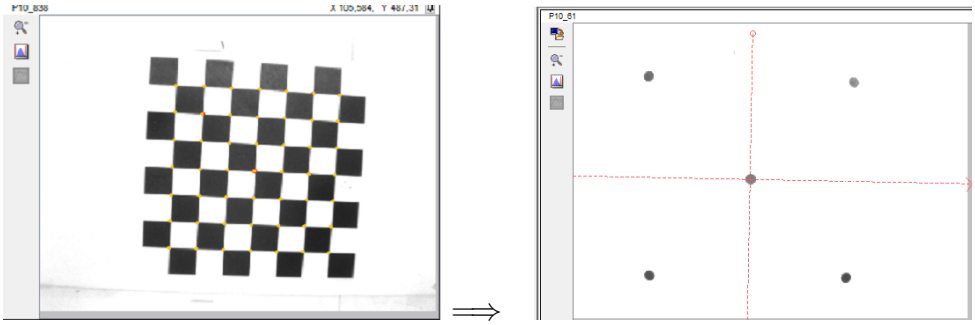
Capitolo 3

Sviluppo di un nuovo strumento di calibrazione

In questo capitolo sarà presentato il progetto di svolto. Partendo da ciò che già era esistente e spiegando i vari tentativi di miglioramento effettuati. Fino ad arrivare alla spiegazione dell'eseguibile finale realizzato.

3.1 Algoritmi Precedenti

Per il raggiungimento del nostro scopo, sono stati analizzati e testati due esistenti "tentativi" di calibrazione, effettuati impiegando il VPM di Impact ed il Laser Engine. Entrambi si basano sull'intervento manuale da parte dell'operatore, il quale deve eseguire una serie di operazioni per calibrare la camera ed ottenere un sistema di riferimento solidale ad un'origine precedentemente scelta.
Uno di questi processi di calibrazione prevede l'impiego del "Target Calibration", scegliendo uno dei tre pattern a disposizione, e successivamente di proiettare i limiti di un oggetto noto. L'utente sarà quindi guidato nel posizionamento di questo oggetto e tramite un apposito tool di rilevamento se ne ricaverà il suo baricentro che diventerà il nuovo sistema di riferimento della camera.
L'altro procedimento di calibrazione, da cui siamo partiti per giungere all'algoritmo finale, prevede l'utilizzo del VPM nella maniera classica: l'uso del tool "Real World

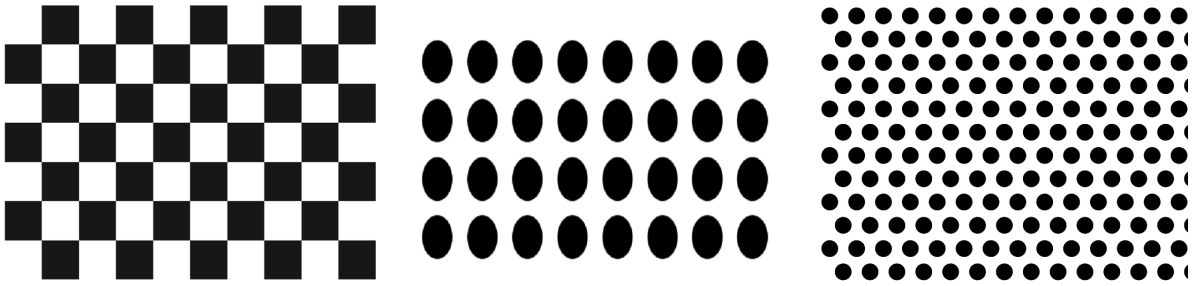


(a) Target Calibration (b) Real World Coordinates

Figura 3.1: Step Calibrazione

Coordinates", in aggiunta al "Target Calibration". In figura 3.1 è possibile vedere i due passaggi effettuati in successione.

Entrambi i pattern sono semplicemente stampati su fogli A4; il primo è composto dalla classica scacchiera, mentre il secondo è formato da 5 cerchi su sfondo bianco. Per eseguire questa procedura si pone innanzitutto la scacchiera all'interno dell'area visibile della camera e si effettua la "Target Calibration". Dopodiché, servendosi del secondo pattern, si cerca di far coincidere il più possibile il cerchio mediano con l'asse centrale di marcatura del laser. A questo punto il tool di Impact "Generates Point" estrapola i baricentri dei blob presenti nella scena. Ad ogni baricentro di ciascun cerchio viene associata, tramite inserimento manuale, la coppia di nuove coordinate di quel cerchio. Infine, si esegue la "Real World Coordinates" che porta al risultato visto in figura 3.1b, in cui si nota il sistema di coordinate finale avente origine nel baricentro del cerchio centrale, come voluto. E' possibile inoltre applicare il tool di "Undistort Image" che corregge l'immagine per ottenere la situazione ideale, la stessa che si sarebbe ottenuta senza la presenza dello "skew" e della distorsione della lente.
Nelle prossime due sezioni saranno illustrati gli studi effettuati per comprendere come eseguire al meglio la calibrazione finale ricercata.



(a) Scacchiera (b) Griglia (c) Esagono

Figura 3.2: Pattern

3.2 Analisi della Target Calibration

La “Target Calibration” rappresenta la parte iniziale dell’algoritmo. Per il suo funzionamento sono stati testati più pattern: scacchiera, griglia, esagonale (in figura 3.2), per capire quale fosse il migliore. Si é accertato che la scacchiera fosse la migliore soluzione poiché garantisce un risultato più stabile e soddisfacente, anche nelle condizioni operative più sfavorevoli. Infatti, l’algoritmo che sta alla base del rilevamento della scacchiera ricerca i punti di incrocio delle righe. Mentre, per quanto riguarda il rilevamento degli altri pattern, viene cercato il baricentro dei vari blob, il quale può risultare non preciso a causa di un elevata inclinazione della camera o in condizioni di luce non omogenee nella scena. Successivamente si è provato a diminuire od aumentare la dimensione dei quadrati della scacchiera per determinare la grandezza ottimale, ma l’esito ottenuto rivela che modificare la grandezza dei quadrati non porta ad un significativo miglioramento o peggioramento del risultato. L’unico vincolo é dato dal numero minimo di colonne e di righe della scacchiera: maggiore o uguale a quattro. Per determinare la corretta o scorretta calibrazione abbiamo fatto riferimento ai due valori, descritti nel primo capitolo, tabella 1.2,: “Maximum value Reprojection Error”(MVRE) e “Average Reprojection Error”(ARE).

Dopo aver studiato il miglior pattern da utilizzare, si è creato un metodo della classe “Camera” che permettesse di effettuare la calibrazione dei parametri intrinseci. Questo metodo, nel codice in 3.1, ha come argomento il valore del “pitch” ed impiega la “porta” di Impact necessaria alla calibrazione sincrona, restituendo l’oggetto ”OperationResult”che contiene le informazioni riguardo al “MVRE” e “ARE” e sulla riuscita

dell'operazione "SnapImageAndCalibrate_Sync()".

```
public OperationResult<VisionCalibrationResult>
   targetCalibrationSync(int pitch)
{
 OperationResult op = new OperationResult();
 OperationResult<VisionCalibrationResult> operationResultCalibr
    = new OperationResult<VisionCalibrationResult>();
 visionDevice.SetRealPortValue_Sync(VisionPort.CreateFromPath("Vision
    System.Camera:Target Pitch"), pitch);
 visionDevice.SetIntegerPortValue_Sync(VisionPort.CreateFromPath("Vision
    System.Camera:Calibration Mode"),
    (int)VisionCalibrationMode.kCheckerboardTarget);
 operationResultCalibr =
    visionDevice.SnapImageAndCalibrate_Sync();
 AfterTargetCalibration(operationResultCalibr);
 return (operationResultCalibr);
}
```

Listing 3.1: Target Calibration in C♯

L'approccio operativo illustrato in questa sezione rimarrà invariato e verrà utilizzato durante il calcolo futuro dei parametri intrinseci.

3.3 Analisi della Real World Coordinates

La maggiore difficoltà riscontrata è quella descritta in seguito, riguardante l'individuazione di uno stesso sistema di riferimento, camera-laser, per il quale sono state testate molteplici soluzioni fondate sul "Real World Coordinates" (WRC). Si è pensato di partire dall'utilizzo del pattern illustrato in figura 3.1b, avente dei cerchi facilmente rilevabili con il metodo di "Generates Point". Un primo miglioramento operativo ha previsto la sovrapposizione dei due pattern, scacchiera e cerchi, stampandoli in un unico

foglio: visibile in figura 3.3. L'operazione di modifica effettuata ha posto al centro di ogni quadrato un cerchio bianco, ed ha aggiunto all'interno di alcuni di essi dei cerchi neri da rilevare. Il vantaggio di questa modifica è rappresentato dal fatto che l'operatore non sarà più obbligato a cambiare il pattern durante l'operazione. I metodi di cali-

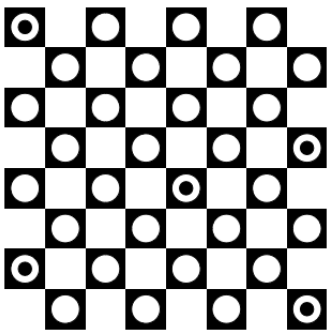


Figura 3.3: Scacchiera Modificata con punti

brazione fin qui illustrati presentano ancora un'accuratezza molto limitata, dovuta sia alla possibile imprecisione del tool di rilevamento, sia ad un fattore umano: un'eccessiva approssimazione da parte dell'utente nel posizionamento dell'oggetto/patternFem può compromettere notevolmente il risultato. Per gestire questo problema si è apportata una modifica sostanziale all'approccio operativo, basata sul fatto che il sistema di riferimento finale è centrato nel mezzo dell'area operativa del laser. Si è pensato quindi di far marcare al laser dei cerchi in delle posizioni da lui note perfettamente, rilevati poi tramite l'utilizzo del "Generates Point". Per questo nuovo metodo di calibrazione si è utilizzato, durante la "Target Calibration", un layout simile a quello della scacchiera modificata, vista precedentemente, eliminando però i cerchi neri all'interno dei quadrati, in figura 3.4. Il layout così ottenuto è stato stampato su un materiale adatto alla marcatura per poter effettuare la seconda operazione. Nella fase successiva è stato introdotto l'uso del laser, che marca dei cerchi di dimensioni millimetriche in posizioni note a priori. Anche con questo procedimento si avrà la necessità di proiettare i limiti della scacchiera, ma poiché la posizione del cerchio da rilevare non è più influenzata dal pattern posizionato, l'operatore sarà più libero da un attento controllo e da una minuziosa precisione. (Tutto ciò è stato possibile perché la dimensione del raggio del cerchio marcato è solamente di pochi millimetri a fronte della maggiore grandezza dei cerchi bianchi, pertanto è realiz-

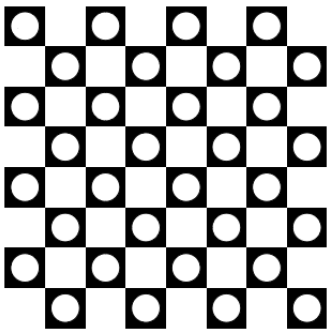
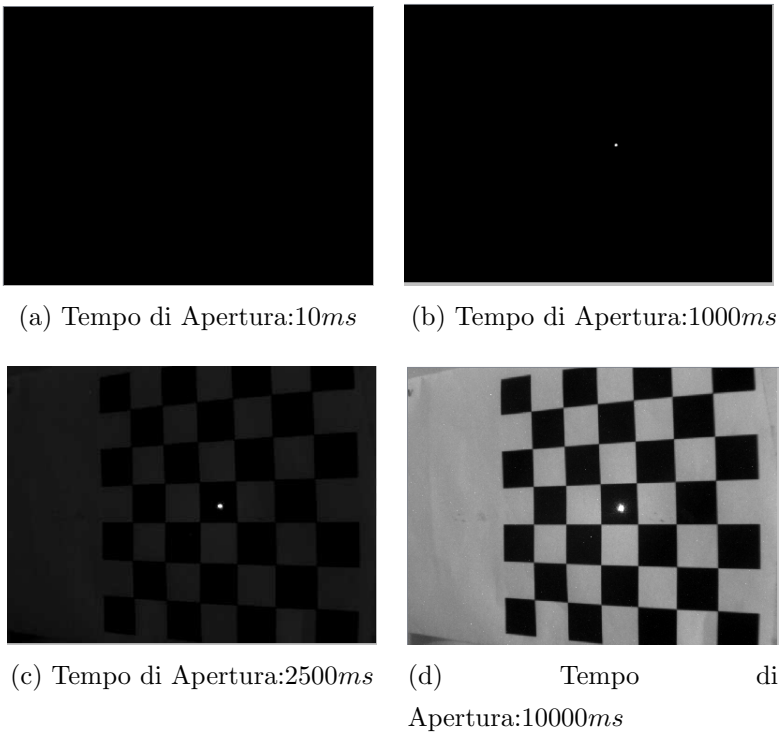


Figura 3.4: Scacchiera Modificata

zabile la marcatura all'interno dello spazio desiderato anche nel caso in cui il pattern non fosse stato posizionato con i dovuti accorgimenti.) Il limite comunque più evidente risulta il fatto che il pattern da marcare è di utilizzo monouso nel caso in cui vengano cambiate le condizioni operative.
Bisogna ricordare che uno degli obiettivi ricercati è stato l'utilizzo del solo Pc Embedded del laser, evitando l'uso di un aggiuntivo pc esterno. Per ottenere questo risultato si è scelto di sostituire il software Impact, fin qui utilizzato, con la più "leggera" libreria "VisionSDK". Infatti Impact risulta un software complesso che richiede un elevato dispendio computazionale, non facilmente sopportabile da un computer embedded che deve svolgere anche altri task. L'uso della libreria "VisionSDK" invece permette sia di integrarsi perfettamente con l'ambiente .NET, che di recuperare i metodi di Impact delle soluzioni precedenti. L'ostacolo più grande è stato il poter riutilizzare il metodo "Generates Point", che determina con ordine sparso i baricentri dei blob, dopo un accurato settaggio della camera. Questa vincolo è facilmente gestibile in VPM, in cui l'utente interagisce direttamente con l'interfaccia grafica, ma è stato difficile da replicare automatizzando l'intero processo. Per poter utilizzare il metodo " Generates Point" in maniera che individuasse nell'ordine corretto i baricentri, è stato necessario introdurre un approccio sequenziale. Infatti la soluzione più semplice da adottare è stata quella di fotografare in successione una scena avente ogni volta un unico oggetto posto di volta in volta in posizioni differenti. A fronte di questo si è pensato di introdurre l'uso di uno dei due diodi laser presenti a bordo di esso, in particolare del diodo simulante la marcatura. Per prima cosa si è studiato come viene osservato dalla camera il fascio luminoso pro-



(a) Tempo di Apertura:10ms (b) Tempo di Apertura:1000ms

(c) Tempo di Apertura:2500ms (d) Tempo di Apertura:10000ms

Figura 3.5: Tempi di Apertura Diaframma

iettato su una qualunque superficie. Bisogna ricordare che la camera utilizzata, P-series 10, supporta solamente la scala di grigi, motivo che ha aumentato la difficoltà del nostro algoritmo. Infatti nel caso contrario, di una camera a colori, avremmo potuto facilmente individuare il punto proiettato essendo di colore rosso. In particolare si è verificato che il settaggio del tempo di apertura dell'otturatore e che l'uso degli illuminatori incide notevolmente sulla resa della fotografia effettuata. Aumentando o diminuendo il tempo di apertura dell'otturatore, il fascio luminoso si presenta in maniera del tutto differente. Partendo da un tempo di apertura praticamente nullo ed andando ad aumentarlo, si noterà il passaggio da un immagine completamente nera ad un immagine avente un singolo punto luminoso. Questo si presenta all'inizio come un punto intermittente ed instabile, non facile da fotografare, poi con l'aumentare del tempo il punto sarà sempre più definito, per poi arrivare infine ad avere contorni non precisi: serie di fotografie in figura 3.5. La situazione di funzionamento ideale nel quale rilevare il punto proiettato è rappresentata dalla figura 3.5b. Con questo settaggio della camera il metodo "Generates

Point" non rischia di andare alla ricerca di punti spuri, ma permette di rilevare il punto in maniera molto precisa. Un ulteriore vantaggio di questa soluzione è rappresentata dal fatto che aprendo l'otturatore per tempi molto brevi, l'unica fonte luminosa che va a colpire il sensore della camera è rappresentata da quella del diodo laser. Per questo motivo questa è la soluzione operativa che si è cercato di replicare attraverso l'eseguibile che verrà descritto nella sezione successiva.

3.4 Applicazione Sviluppata

In questa sezione si illustrerà nel dettaglio l'eseguibile creato, in figura 3.6, e i file di cui necessita per un corretto funzionamento. Il sistema finale è composto dall'eseguibile principale scritto in C#, dal file Javascript "Ethernet Protocol" che permette di interpretare i comandi operativi inviati al laser, e da due file:"sacchiera.xlp" e "point.xlp" del tutto invisibile all'utente. Quest'ultimo dovrà per prima cosa lanciare il file javascript e successivamente l'eseguibile per la calibrazione, entrambi presenti nel pc Embedded. Si è

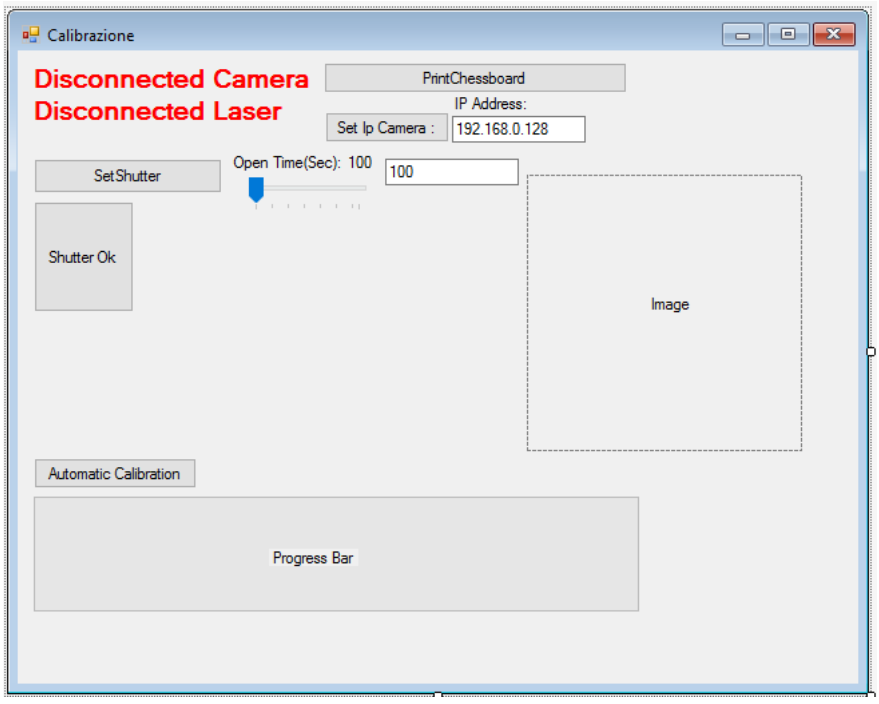


Figura 3.6: Interfaccia eseguibile

scelto di automatizzare l'ultimo procedimento operativo spiegato nella sezione precedente: cioè l'utilizzo del "Target Calibration" con la scacchiera, seguito dall'uso del diodo laser per effettuare la "Real World Coordinates". L'eseguibile si fonda su Windows Form che ha permesso di creare una semplice interfaccia grafica con cui l'utente può interagire. I pulsanti, visibili nella figura 3.6, vengono visualizzati in successione, in questo modo l'utente sarà guidato nelle varie fasi operative dell'algoritmo tramite la visualizzazione del pulsante corrispondente ad una particolare azione. Si procederà di seguito con la spiegazione dei passaggi operativi effettuati.
L'eseguibile realizzato permette l'apertura di un file "PDF" contenente il pattern della scacchiera, da stampare su un semplice foglio A4. E' stata utilizzata una scacchiera avente i lati dei quadrati di lunghezza 14 mm; dato che identifica il "pitch" durante la calibrazione dei parametri intrinseci. E' possibile anche scegliere l'indirizzo IP della camera, se non è stato cambiato è di default il: "192.168.0.128". Inoltre è stato gestito l'uso dei diodi laser, in particolare si è utilizzato il diodo centrale per simulare la marcatura dei limiti della scacchiera da posizionare. E' stato indispensabile disabilitare il laser laterale durante l'intero algoritmo poiché interferisce in maniera negativa con la scena, infatti la presenza di un fascio luminoso esterno appare come una notevole fonte di disturbo per la camera. Per i motivi illustrati, il JavaScript, nel momento della sua esecuzione, disabilita immediatamente il diodo in questione e lo riattiva solamente al termine dell'operazione di calibrazione. Si ricorda che durante l'intero processo di calibrazione non verrà mai attivato il laser vero e proprio che permette la marcatura.
L'utente, dopo aver stampato la scacchiera, dovrà posizionarla all'interno dell'area di lavoro del laser, zona in cui vogliamo garantire sia la correzione prospettica che della lente. Per facilitare questo posizionamento viene impiegato il diodo laser simulante la ripetuta marcatura di un quadrato avente le dimensioni della scacchiera. Cosi facendo l'operatore vedrà proiettato un quadrato luminoso di colore rosso all'interno dell'area di lavoro del laser. Per questa operazione è stato creato un file contenente un semplice quadrato, "scacchiera.xlp", ed è stato caricato in memoria attraverso il metodo: 'public byte[] CreateMessageOpenDocumentFromDevice(String path)", che permette di creare un messaggio per selezionare il pattern da marcare. Per poter far riferimento al file però esso deve essere già presente nel seguente path "Lighter Data\Docs\Layouts" del Pc

embedded del laser. Per questo motivo, al momento dell'avvio dell'eseguibile vengono spostati entrambi i file .xlp, precedentemente creati, all'interno della cartella designata; al termine della calibrazione questi file vengono prontamente eliminati per non lasciar alcuna traccia del processo all'operatore. L'utente, dopo aver posizionato il pattern all'interno di questi limiti proiettati, potrà procedere con l'operazione successiva: la "Target Calibration". Lo sftep successivo prevede un attento e preciso settaggio del tempo di apertura dell'otturatore, poiché per una buona calibrazione è di fondamentale importanza la corretta visibilità della scacchiera. Per questo è stata inserita nell'interfaccia grafica la possibilità, da parte dell'utente, di diminuire od aumentare questo tempo, ed in più di vedere l'immagine scattata cliccando su "Set Shutter". In questo modo l'operatore sarà in grado di comprendere quando la scena risulta ben nitida, avendo a disposizione all'interno dell'eseguibile la fotografia del pattern. Per un miglior funzionamento sono stati anche abilitati gli illuminatori della camera, i quali permettono di ridurre le fonti di rumore presente nell'immagine. Dopo aver settato la camera con questi semplici accorgimenti si può procedere alla calibrazione dei parametri intrinseci. Si è realizzato il metodo, già visto nella sezione: 3.2, che restituisce come risultato i valori della "MVRE" e della "ARE". Comparandoli con i valori presenti nella tabella 4.1 del capitolo 2: è possibile capire se si è effettuata una corretta calibrazione.
Al termine della "Target Calibration", cliccando sul pulsante "Automatic Calibration" la procedura diventa del tutto autonoma e non si prevede più l'intervento dell'operatore. Durante l'intero processo gli illuminatori della camera vengono disabilitati, per non interferire con la scena. Lo scopo finale è quello di utilizzare la "Real World Coordinates", avendo a disposizione le coordinate dei baricentri dei cerchi espresse in entrambi i sistemi di riferimento: quello attuale della camera ed il nuovo. Il principio di funzionamento di questo algoritmo si basa sullo spostamento di un cerchio luminoso in posizioni da noi note e acquisendone il baricentro. Un corretto funzionamento prevede il rilevamento delle coordinate di almeno 4 punti. Nel nostro caso si sono presi in considerazione 7 cerchi, aventi coordinate rispetto al sistema di riferimento laser pari a: (0,0);(5,0);(0,-5);(35,0);(0,-35);(-35,0);(-35,-35); ed archiviati in un vettore di Point:"pointsPrint". Per poter ottenere questo cerchio luminoso, è stato realizzato un file in Lighter, "point.xlp", formato da un cerchio pieno dalle dimensioni di $4mm$ identificato da un particolare id:

"1". L'uso di questo id è stato necessario per poter identificare l'oggetto in Lighter da dover spostare. Per la creazione del messaggio da inviare al Javascript si è utilizzato il metodo della classe Laser:"public byte[] CreateMessageMoveDataFieldPoint(int iD, Point point, Double Angle)", che permette lo spostamento dell'oggetto nelle coordinate desiderate.
La procedura principale da ripetere ciclicamente è stata quindi quella di spostare il punto nelle posizioni note e di rilevarne il baricentro, per questo è stato utilizzato un ciclo For. Il codice è visibile in 3.2, ora si analizzeranno le istruzioni che lo compongono.

```
for (int i = 0; i < 7; i++)
{
  message =
     laser.CreateMessageMoveDataFieldPoint(pointsPrint[i]);
  laser.SendMessage(message);
  laser.Mark();
  System.Threading.Thread.Sleep(1500);
  Application.DoEvents();
  foundPoint = camera.FixShutterAndFindPoint();
  imagePointList.Add(foundPoint);
  laser.StopMarking();
}
```

Listing 3.2: Ciclo For

Ad ogni iterazione si seleziona la coordinata del cerchio da marcare prelevandola dal vettore: "pointsPrint", precedentemente creato, e si invia il comando al laser per la simulazione di marcatura. Tra quest'ultima istruzione e la rilevazione del baricentro del cerchio è stato introdotto un Thread di "Sleep" della durata di un secondo, per essere sicuri che il laser avesse il tempo necessario per spostare il punto da marcare. A questo punto si utilizza il metodo principale dell'intero algoritmo: "FixShutterAndFindPoint()". Questo metodo della classe camera, visibile in 3.3, è stato creato per gestire il problema già illustrato derivante dal "Generates Point".

```
public VisionPoint FixShutterAndFindPoint()
{
 VisionPoint foundPoint = new VisionPoint();
 VisionPoint meanFoundPoint = new VisionPoint();
 int iteration = 0, shutter = 100;
 double meanX = 0, meanY = 0;
 ShutterSpeed(shutter);
 List<VisionPoint> pointList = new List<VisionPoint>();
 pointList = GeneratesPointList();
 while (iteration < 20)
 {
   while (pointList.Count != 1)
   {
    shutter += +100;
    Console.WriteLine("No Point Found");
    iteration = 0;
    meanX = 0;
    meanY = 0;
    this.ShutterSpeed(shutter);
    pointList = GeneratesPointList();
   }
   meanFoundPoint = pointList[pointList.Count - 1];
   meanX = meanFoundPoint.X + meanX;
   meanY = meanFoundPoint.Y + meanY;
   pointList = GeneratesPointList();
   iteration++;
  }
  meanX = Math.Round((meanX/iteration), 3);
  meanY = Math.Round((meanY/iteration), 3);
  foundPoint = VisionPoint.Create(meanX, meanY);
  return foundPoint;
}
```

Listing 3.3: Individuazione del Cerchio

Grazie allo studio eseguito in precedenza, riassunto nella figura 3.5, che mostra come varia la visibilità del cerchio in base al tempo di apertura dell'otturatore, è stato possibile ideare il seguente algoritmo. Si è quindi inizializzato l'algoritmo ad un tempo molto basso, pari a 100 ms, con il risultato di un immagine completamente nera. Dopodiché, aumentando questo tempo con incrementi di 100 ms, l'immagine introduce gradatamente un piccolo punto luminoso. Attraverso l'uso del "Generates Point" è possibile rilevare il baricentro dell'unica figura presente nella scena. Lo svantaggio principale è dettato dal fatto che per alcuni valori, ancora troppo bassi del tempo di apertura, il cerchio risulta come un punto luminoso fortemente instabile. Per questo motivo si decide di considerare valida la coordinata solo del punto che rimane permanente, e si scartano i baricentri derivanti da un punto instabile. La precisione della misura è stata migliorata calcolandone il valor medio, derivato da 20 misure ripetute dello stesso cerchio. Per effettuare quest'operazione è stato creato un ciclo "while", , con un numero di iterazioni pari a 20. Si rimane in questo ciclo fin quando viene rilevato per 20 volte consecutive il baricentro dello stesso cerchio luminoso mantenendo invariato il tempo di apertura; in casi di punti ancora con visibilità instabile, il ciclo viene reinizializzato. Cosi facendo il punto viene definito stabile solamente quando viene visualizzato per 20 volte consecutive, mantenendo fisso il tempo di apertura . All'interno di questo ciclo è presente un ulteriore ciclo "while", che termina solamente nel caso in cui viene rilevato un baricentro nella scena. Esso, si ripete fin quando il metodo "Generates Point" restituisce un numero di punti pari a 0, a questo punto si incrementa il tempo di apertura dell'otturatore di 100 ms e viene reinizializzato il ciclo "while" più esterno. Entrambi i cicli terminano solamente quando avremo ottenuto per 20 volte un baricentro stabile. Attraverso queste differenti misure, riferite allo stesso cerchio, si calcola il valor medio, sia per la coordinata x che per la coordinata y. Al termine di questo metodo, "FixShutterAndFindPoint()", è disponibile un punto che viene aggiunto alla lista ordinata dei punti immagine: "imagePointList". Infine viene interrotta la marcatura per riprendere di nuovo l'inizio del ciclo For. Con la procedura descritta è stato possibile archiviare in maniera sequenziale le coordinate

immagine dei baricentri, avendo a disposizione anche le corrispondenti coordinate nel nuovo sistema di riferimento. In questo modo si ottengono cosi due liste di punti che servono all'esecuzione del metodo "Real World Coordinates". Anche in questo caso, come "opearation result", vengono restituiti i valori dell "MVRE" e della "ARE".

Capitolo 4

Risultati e Conclusioni

L'eseguibile descritto ha apportato notevoli vantaggi operativi ed ha permesso di semplificare l'integrazione della camera con il laser. Grazie all'uso del diodo laser è stato possibile ridurre il numero di settaggi eseguiti dall'operatore, sfruttando il fatto che, nella seconda parte dell'algoritmo, tutto diventa automatizzato, ignorando completamente le condizioni di luce presenti nella scena. E' stato effettuato uno studio statistico per comprendere l'efficienza dell'algoritmo. E' stato testata la precisione del metodo "FixShutterAndFindPoint()", rilevando, per 100 volte consecutive, i baricentri di 20 punti sparsi nella scena. Questo procedimento è stato eseguito dopo aver già calibrato l'intero sistema, avendo quindi il piano immagine già scalato rispetto i mm e avente il sistema di riferimento centrato nell'area di lavoro del laser. Simulando la marcatura di un cerchio in una determinata coordinata, si è voluto misurare la differenza fra il valore, ottenuto tramite il metodo "FixShutterAndFindPoint()", e la coordinata reale del punto. Nella tabella 4.1 è possibile vedere l'errore medio e l'errore massimo dei punti misurati. Fissato il punto reale l'errore viene calcolato come la differenza di questa coordinata dalla coordinata misurata. E' possibile vedere come l'errore massimo commesso sia maggiore nei punti più lontani dal centro dell'asse ottico, ciò è dovuto all'effetto di distorsione della lente. Questo errore risulta comunque molto contenuto infatti si parla al massimo di 0,5 millimetri.

Per comprendere meglio i futuri sviluppi applicativi, derivanti dalla buona riuscita dell'eseguibile, è stato provato anche un possibile funzionamento del sistema finale: camera

Coordinata Punto Reale	Errore medio X	Errore medio Y	Errore Max X	Errore Max Y
(5,5)	0,07	0,07	0,2	0,2
(5,-5)	0,07	0,04	0,1	0,1
(-5,5)	0,03	0,04	0,1	0,1
(-5,-5)	0,03	0,02	0,08	0,09
(10,10)	0,09	0,05	0,2	0,1
(10,-10)	0,04	0,02	0,1	0,09
(-10,10)	0,06	0,08	0,1	0,2
(-10,-10)	0,12	0,05	0,3	0,1
(20,20)	0,18	0,02	0,3	0,07
(20,-20)	0,03	0,05	0,07	0,1
(-20,20)	0,09	0,11	0,2	0,2
(-20,-20)	0,11	0,08	0,2	0,2
(30,30)	0,09	0,05	0,2	0,1
(30,-30)	0,03	0,06	0,1	0,1
(-30,30)	0,18	0,07	0,3	0,2
(-30,-30)	0,14	0,07	0,2	0,2
(40,40)	0,2	0,2	0,4	0,3
(40,-40)	0,1	0,1	0,3	0,2
(-40,40)	0,3	0,04	**0,5**	0,3
(-40,-40)	0,3	0,04	0,4	0,1

Tabella 4.1: `Perfomance del metodo ‘‘FixShutterAndFindPoint()’’`

più laser. Uno dei più interessanti risultati è l'utilizzo della camera per individuare il baricentro di un oggetto nella scena, e comunicarne le coordinate al laser. Grazie alla calibrazione effettuata sarà possibile rototraslare opportunamente il file da marcare nella posizione centrale dell'oggetto. Si è creato, tramite VPM, un particolare file di ispezione che si serve del tool "undistort image" seguito dal "blob detection". Quest'ultimo tool permette di rilevare nella scena la presenza di un "blob" fissando manualmente un livello di threshold (i pixel avente un intensità maggiore del valore scelto andranno a comporre il "blob"). Per migliorare la misurazione è possible anche indicare l'area minima e massima del "blob" che si vuole rilevare. Tutto ciò è stato applicato nell'esempio visibile in figura: 4.1. Infine è stato creato il metodo: "public byte[] CreateMessageMo-

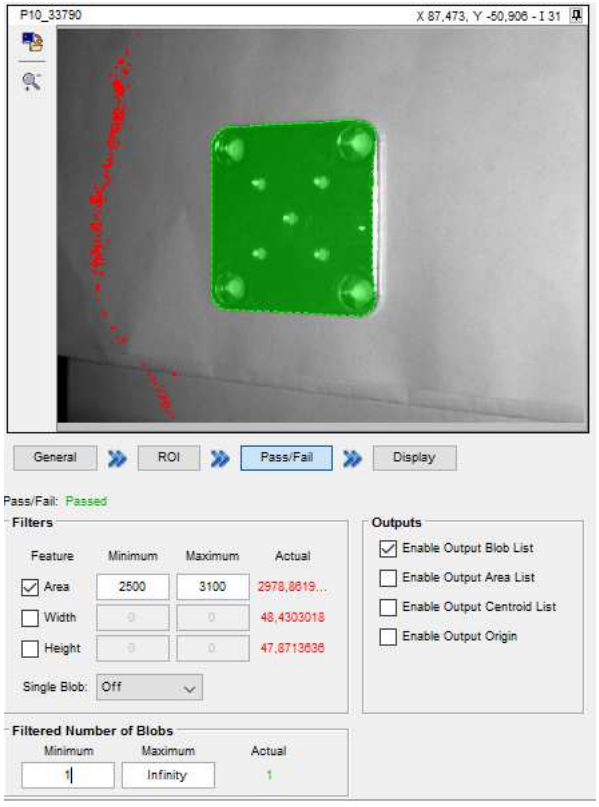


Figura 4.1: Blob Detection

veDataFieldPoint(int iD, Point point, Double Angle)" della classe Laser che permette la rototraslazione dell'intero documento da marcare; cosi facendo sarà possibile marcare il documento avente l'origine nel baricentro dell'oggetto. Un tool sostitutivo, anche

più preciso, al "blob detection" è il "pinpoint pattern tool", che permette di effettuare un apprendimento off-line dell'oggetto da ricercare. In conclusione, uno tra i possibili miglioramenti implementativi del sistema integrato camera-laser prevede solamente l'utilizzo di Lighter per applicazioni di visione relativamente semplici. L'idea futura è quella di aggiungere come sfondo al layout da stampare, l'immagine vista dalla camera; in questa maniera l'operatore potrà definire perfettamente dove e come marcare il layout creato, avendo un immediato riscontro sull'oggetto reale, visibile grazie alla camera. Si vorrebbero anche aggiungere dei semplici tool di rilevamento dell'oggetto, come ad esempio "blob detection", per evitare il settaggio della camera tramite VPM, posto su un pc esterno. Nel caso in cui però l'utente dovrà effettuare delle complicate operazioni di visione, sarà necessario l'impiego di quest'ultimo.

Bibliografia

[1] P-Series Reference Manual 821003525 (Rev E.) Ed.: 010/2015.

[2] P-Series Hardware Guide en Rev E.

[3] UniQ User's Manual Ed:821003145.

[4] Lighter Laser Editor Ita (rev.6.2.0).

[5] CameraModelCalibrationRectification.pdf - T3lab.

[6] Z. Zhang, "A flexible new technique for camera calibration", IEEE Trans. On Pattern Analysis and Machine Intelligence, 22(11):1330-1334, November 2000.

[7] `http://www.cs.unibo.it/~ghini/didattica/reti_lpr/TAC006a.pdf`

[8] Protocollo di rete. (20 settembre 2015). Wikipedia, L'enciclopedia libera. Tratto il 19 agosto 2016, 08:47 da `it.wikipedia.org/w/index.php?title=Protocollo_di_rete&oldid=75284366`

[9] `https://msdn.microsoft.com/it-it/library/b6xa24z5(v=vs.110).aspx` Developer Network C#

[10] "Lighter TcpServer.pdf"

Ringraziamenti

Desidero ringraziare tutti coloro che mi hanno aiutato nella realizzazione della mia tesi. Ringrazio il mio Relatore Prof. Di Stefano e correlatore Ing.Imbriaco per il tempo dedicatomi. Ringrazio la mia famiglia per il grandissimo supporto morale, psicologico ed anche finanziario. Ringrazio la mia Bea per essermi stata sempre vicino in ogni momento. Ringrazio i miei amici per aver fatto ritardare il più possibile questa data. Ringrazio i miei colleghi di corso, nonché grandi amici, che mi hanno aiutato ed accolto. Ringrazio tutte le persone conosciute durante questi lunghi anni, che mi hanno permesso di crescere molto come persona e di giungere a questo importantissimo traguardo.