POLITECNICO DI MILANO

Facoltà di Ingegneria dei Processi Industriali

Corso di Laurea Magistrale in Ingegneria Elettrica

Dipartimento di Elettrotecnica

FINGERTIP 3D POSITION TRACKING USING STEREO VISION ALGORITHMS

Relatore: Prof. Maria Stefania Carmeli

Correlatore: Prof. Kenji Kawashima

Tesi di Laurea Specialistica
di:
Davide Cavaliere
Matr. 749923

Anno Accademico 2010-2011

To my mum, dad and Paolo.

“not everything that can be counted counts and not everything that counts can be counted” (Albert Einstein).

Contents

1 Prefazione **1**

2 Introduction **8**

3 Stereo vision experimental setup **13**
- 3.1 Stereo vision system . . . 13
- 3.2 Software . . . 14

4 Image acquisition and processing **17**
- 4.1 Image data acquisition . . . 17
  - 4.1.1 Camera image acquisition . . . 18
  - 4.1.2 RGB and HSV color spaces . . . 20
  - 4.1.3 Exposure, saturation and white balance . . . 24
  - 4.1.4 Image noise filtering . . . 27
- 4.2 Image processing algorithms . . . 29
  - 4.2.1 Color probability histograms . . . 30
  - 4.2.2 Back-projection . . . 32
  - 4.2.3 Image thresholding . . . 33
  - 4.2.4 Erode and dilate filters . . . 34
  - 4.2.5 Contours retrieving . . . 37
  - 4.2.6 Contours smoothing . . . 38
  - 4.2.7 Fingertips search algorithm . . . 39
  - 4.2.8 Distance calculation using stereo vision . . . 42
  - 4.2.9 Fingertips space matching and noise filtering . . . 45
  - 4.2.10 Fingertips time matching and noise filtering . . . 48

5 Project analysis and results **50**
- 5.1 Program analysis . . . 50

5.1.1 Program structure 50
5.1.2 Frame acquisition and camera adjust 56
5.1.3 Histogram creation 59
5.1.4 Fingertip retrieving 60
5.1.5 Fingertip matching 64
5.1.6 Graphical user interface 68
5.2 Results 75
5.2.1 Fingertip standing still 76
5.2.2 Fingertip rotating 82

6 Pros/cons and problems encountered 88
6.1 Pros and cons 88
6.2 Problems encountered 91

7 Conclusion and future improvements 94
7.1 Conclusion 94
7.2 Future improvements 95

Bibliography 100

Thanks 101

List of Figures

1.1 Robot chirurgico del Kagawa-Kawashima Lab. 2
1.2 NDI Polaris Spectra Hybrid. 6
1.3 Finestre del programma. 7

2.1 Kagawa-Kawashima Lab surgery robot. 9
2.2 NDI Polaris Spectra Hybrid system. 12

3.1 Stereo vision system. 14
3.2 Program windows. 15

4.1 Bayer filter on camera sensor. 19
4.2 RGB color functions. 21
4.3 RGB image decomposition. 22
4.4 HSV cylinder. 23
4.5 Gamut. 25
4.6 Lamp spectrum. 26
4.7 Image noises. 27
4.8 Mean filters. 28
4.9 Example of mean filter. 29
4.10 RGB Histograms. 30
4.11 Histogram reference images. 31
4.12 Example of 2-dimensional Hue Saturation histogram. 32
4.13 Back-projection image. 33
4.14 Thresholded image. 34
4.15 Erode filter. 35
4.16 Dilate filter. 36
4.17 Contour retrieving. 37
4.18 Contour smoothing. 38

4.19 Fingertips search example. 39
4.20 Vectors in fingertips search algorithm. 40
4.21 Found fingertips. 41
4.22 Hand with found fingertips. 42
4.23 Camera axis, upper plane. 43
4.24 Camera axis, lateral plane. 43
4.25 Stereo vision 3D calculation on x-axis and y-axis. 44
4.26 Stereo vision 3D calculation on z-axis. 45
4.27 Fingertips false detects. 46
4.28 Fingertips matches after spacial noise filtering. 47

5.1 Software structure. 55
5.2 Camera parameters setting. 57
5.3 Histogram left and right reference images. 60
5.4 Histogram left and right mask images. 60
5.5 Histogram hand area selection. 61
5.6 Back-projection calculation. 62
5.7 Hand area. 63
5.8 Fingertips not filtered. 64
5.9 Fingertips filtered. 66
5.10 Program with regulations. 69
5.11 Program with all windows. 71
5.12 Manual selection of the hand area. 75
5.13 Fingertip standing still. 77
5.14 Standing finger position log. 78
5.15 Filters transfer functions. 79
5.16 Position in the three axis using the “2 filter” (standing). . . . 80
5.17 Position in the three axis using the “10 filter” (standing). . . 81
5.18 Fingertip standing 3D position with “2 filter”. 81
5.19 Fingertip standing 3D position with “10 filter”. 82
5.20 Rotating finger position log. 83
5.21 Position in the three axis using the “2 filter” (rotating). . . . 85
5.22 Position in the three axis using the “10 filter” (rotating). . . . 86
5.23 Fingertip rotating 3D position with “2 filter”. 86
5.24 Fingertip rotating 3D position with “10 filter”. 87

6.1 Fingertip occlusion. 92

7.1 Microsoft Kinect. 96
7.2 Kinect block diagram. 97

List of Tables

1.1 Confronto tra diversi tipi di interfacce uomo-macchina. 5

2.1 Comparison between different kinds of human-machine interfaces. 11

Chapter 1

Prefazione

Negli ultimi anni molti diversi tipi di interfacce uomo-macchina sono stati sviluppati dagli ingegneri, in modo tale da permettere ad un calcolatore di comprendere i movimenti desiderati dall'utente. Infatti, spesso capita che nelle applicazioni ingegneristiche sia necessario controllare la posizione di un sistema basandosi sui comandi di un utente. Ci sono molti tipi di interfacce uomo-macchina, ma tutte possono essere suddivise in due macrocategorie [1].

- Interfacce uomo-macchina che richiedano un contatto fisico. Questa tipologia di interfaccia è la più diffusa. Quando la tecnologia ha cominciato a svilupparsi, le interfacce richiedevano un contatto fisico da parte dell'utente. Per esempio, l'utente aveva bisogno di premere un tasto, muovere una leva, girare una manopola, etc. Negli ultimi anni, nuove interfacce hanno cominciato ad apparire sul mercato: esse sono costituite da joystick, joypad oppure sistemi simili poiché sono intuitivi; tutte queste interfacce richiedono che l'utente muova o prema qualcosa, quindi richiedono tutte un contatto fisico. In Figura 1.1 viene mostrato un esempio di interfaccia che richiede un contatto fisico: è un robot chirurgico costruito nel laboratorio Kagawa-Kawashima Lab del Tokyo Institute of Technology.

- Interfacce uomo-macchina che non richiedano un contatto fisico. Esse non richiedono che l'utente interagisca direttamente con l'interfaccia poichè essa non viene fisicamente toccata. Ad esempio, il riconoscimento vocale e la computer vision appartengono a questa categoria. In questo caso vengono utilizzati microfoni, fotocamere e altre tipologie di sensori.

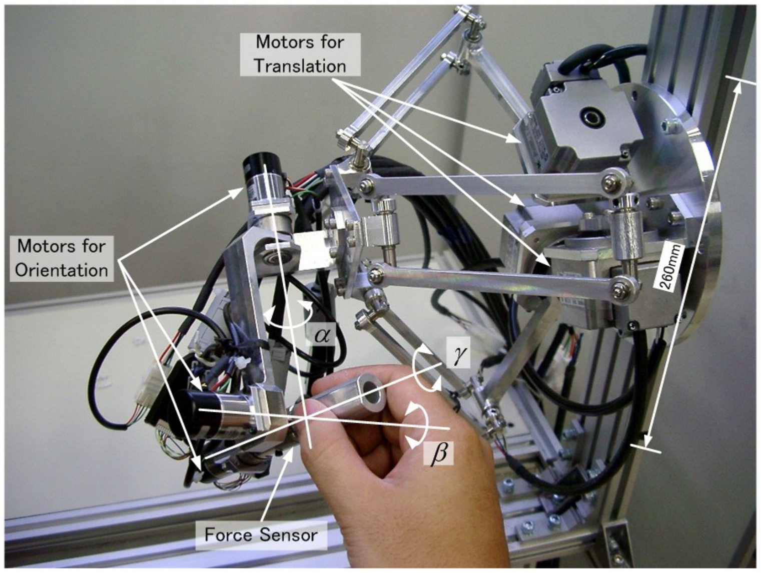


Figure 1.1: Robot chirurgico del Kagawa-Kawashima Lab.

Entrambi i tipi hanno pregi e difetti. A volte, quando l'utente necessita di controllare un attuatore, potrebbe essere molto utile fare ciò senza il bisogno di avere un contatto fisico. Per esempio, potrebbe essere una buona cosa comandare alcune funzioni semplicemente mediante l'utilizzo della voce oppure muovendo una specifica parte del corpo come ad esempio la mano oppure il braccio [2]. D'altra parte, questa nuova tipologia di interfacce a volte può non essere completamente affidabile per diversi motivi; per esempio, le interfacce che si basano su segnali audio non sono in grado di funzionare se l'ambiente in cui si trovano è rumoroso ed inoltre i segnali vocali cambiano a seconda del tipo di voce dell'utente e delle caratteristiche ambientali. Le interfacce video basate sulle fotocamere hanno simili problemi: sono sensibili alle condizioni ambientali tra cui le condizioni luminose e il colore degli oggetti; oltre a ciò, possono presentarsi anche problemi relativi ad occlusioni, ad esempio nel caso in cui un oggetto aggiuntivo si trovi ad essere interposto tra la fotocamera e l'utente.

Il presente lavoro di tesi è stato realizzato presso il Kagawa-Kawashima Lab dell'università giapponese Tokyo Institute of Technology. Il laboratorio in questione si occupa di sistemi pneumatici e del loro utilizzo come attuatori in diverse applicazioni. In particolare, il Prof. Kawashima porta avanti da qualche anno un progetto riguardante la telechirurgia avente lo scopo di permettere ai chirurghi di eseguire operazioni a distanza. Al momento, l'interfaccia con il chirurgo è del primo tipo, ossia richiede al chirurgo di muovere con ciascuna mano un joystick a sei gradi di libertà. Questo tipo di interfaccia, seppure affidabile, non è molto intuitivo e si è deciso dunque di ricercare una nuova soluzione che permetta di riconoscere la posizione delle dita delle mani mediante un metodo che non richieda un contatto fisico. Più specificatamente, si è deciso di optare per un sistema basato sulla visione stereoscopica (stereo vision) in grado di determinare la posizione delle dita nello spazio e di poterli quindi attuare a distanza. La tesi ha lo scopo dunque di mostrare al lettore la soluzione proposta costituita da un sistema di due fotocamere (stereo vision) collegate ad un calcolatore su cui viene fatto funzionare un programma creato dall'autore: il software ha lo scopo di trattare i dati provenienti dalle due fotocamere per poter determinare, istante per istante, la posizione di almeno un dito nello spazio. Questa soluzione potrebbe poi essere estesa anche ad altre tipologie di mercato che non riguardino per forza il settore medicale.

A volte è molto difficile scegliere tra interfacce che richiedano o meno un contatto fisico. Da una parte, le prime sono più affidabili ma richiedono di essere toccate, quindi richiedono la vicinanza dell'utente; dall'altra parte, le seconde sono più intuitive e naturali poiché non richiedono che l'utente tocchi alcun apparato, tuttavia presentano ancora molti problemi sul piano dell'affidabilità. Nel futuro, probabilmente la tecnologia preferirà il secondo tipo - interfacce che non richiedano contatto fisico - poiché esse si avvicinano maggiormente al comportamento umano. Infatti, quando gli esseri umani interagiscono tra di loro, la gran parte delle informazioni vengono scambiate tra gli individui utilizzando la voce e i movimenti del corpo, senza alcun contatto fisico tra le persone. Tuttavia, per poter fare ciò, gli ingegneri devono fronteggiare ancora molti problemi riguardanti l'affidabilità e la stabilità; infatti, una nuova interfaccia può essere utilizzata nelle applicazioni reali solamente se è in grado di soddisfare particolari condizioni di affidabilità. Ciò è dovuto al fatto che un'interfaccia che deve lavorare in un ambiente critico necessita di essere affidabile altrimenti, in caso di disfunzioni, può causare danni a persone oppure una perdita economica. Per questo molti ingegneri e scienziati si stanno orientando su queste nuove interfacce cercando di aumentarne l'affidabilità. Molti buoni risultati sono già stati raggiunti nel settore dei videogiochi e dell'intrattenimento dato che in queste aree i requisiti di affidabilità non sono molto elevati. Tuttavia, nel settore industriale e in altri settori speciali, l'affidabilità è il requisito più importante ed è per questo motivo che questa nuova tipologia di interfacce fa ancora fatica ad imporsi sul mercato. Molta ricerca è stata fatta per cercare di capire il comportamento gestuale umano: l'approccio può essere basato semplicemente sull'utilizzo di una fotocamera, nel caso in cui l'unico scopo sia quello di capire la tipologia di gesto [3] e non sia necessario capire la posizione della mano nelle tre dimensioni [4]. Capire la tipologia di gesto può essere utile perché ogni gesto umano può essere associato ad un particolare ordine impartito dall'utente [5]. Tra le parti più importanti del corpo umano per dare dei comandi ci sono le dita: molti ricerche sono state orientate a cercare la posizione della punta delle dita in un'immagine, combinando anche diverse tipologie di algoritmi per aumentare la stabilità del metodo [6]. Inoltre, alcuni studi sono stati fatti per isolare le aree relative alla pelle umana in un'immagine, grazie ad algoritmi che si basano sul colore [7]. Altre ricerche sono invece orientate alla visione stereoscopica (stereo vision): utilizzando due fotocamere digitali è possibile calcolare la distanza di un oggetto in uno spazio tridimensionale [8] oppure realizzare una ricostruzione tridimensionale della forma di un oggetto [9]. La posizione della punta delle dita può essere calcolata utilizzando un modello tridimensionale della mano [10], ma un'altra tecnica può essere quella di valutare le informazioni sulla curvatura del contorno relativo alla mano [11]. In questa tesi sono state utilizzate le informazioni sulla curvatura del contorno della mano; inoltre, alcuni filtri innovativi sono stati implementati per filtrare i "noisy fingertips".

In questa tesi, il riconoscimento dell'oggetto in questione viene fatto utilizzando la "stereo vision" cioè mediante l'utilizzo di due fotocamere. Più specificatamente, il programma realizzato è in grado di tracciare la posizione del dito indice mentre si muove nello spazio. Questo sistema può essere utile in molti campi che richiedano delle "haptic interfaces", ossia interfacce in grado di riprodurre dei movimenti a distanza. Il programma riconosce la posizione del dito (in tre coordinate); questa posizione può essere inviata ad un attuatore in grado di riprodurre a distanza la stessa posizione spaziale. Un possibile scenario è il seguente: l'utente vuole spostare un carico pesante in un impianto di produzione industriale. Ciò può essere fatto utilizzando un'interfaccia del primo tipo (che richieda un contatto fisico), per esempio un joystick con dei pulsanti; tuttavia è possibile fare ciò anche calcolando la posizione delle dita dell'utente, in modo tale che egli non abbia da fare altro che muovere il proprio dito in aria; il computer deve quindi essere in grado di capire la posizione nei tre assi x, y e z. Quando la posizione viene calcolata, viene inviata ad un attuatore utilizzando un protocollo di comu-

	Kinect	NDI Polaris	Stereo vision
Markers	no markers	passivi/attivi	no markers
Applicazione	intrattenimento	medicale	intrattenimento, industriale
Risoluzione	bassa (3 mm)	alta (0.5 mm)	bassa (2 mm)
Prezzo	basso	alto	molto basso
Affidabilità	alta	alta	deve essere incrementata

Table 1.1: Confronto tra diversi tipi di interfacce uomo-macchina.

nicazione e può quindi essere attuata a distanza. Al momento, sul mercato, alcune interfacce possono soddisfare questi requisiti. Nella tabella 1.1 è stato fatto un confronto tra diverse tecnologie di tracciamento della posizione. La tecnologia Kinect è utilizzata nella consolle di gioco Microsoft X-Box: essa integra una telecamera con sensore RGB ed un sensore di distanza ad infrarossi; la sua risoluzione in termini di distanza non è molto alta (3 mm) ma il costo è basso perché questa tecnologia è utilizzata nel settore videoludico. NDI Polaris (Figura 1.2) è un sistema di tracciamento della posizione usato nel settore medicale per tracciare la posizione di alcuni “markers”; la sua risoluzione è elevata ma anche il costo è alto, ed inoltre viene richiesta la presenza di alcuni markers indossati dall’utente. La tecnologia proposta, basata sulla stereo vision, può essere applicata in campi che richiedano un basso costo: infatti, il sistema è molto economico dato che non viene richiesto un sensore di distanza addizionale. Alcuni possibili settori target sono quello dell’intrattenimento e, nel caso in cui l’affidabilità venga incrementata, anche il settore industriale. La risoluzione spaziale ottenibile dipende dalla risoluzione delle fotocamere.

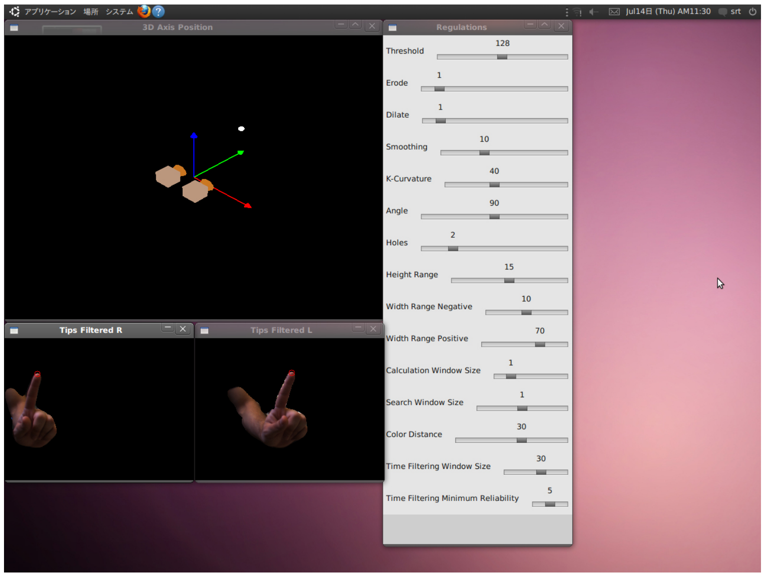
Questo progetto è stato sviluppato combinando diverse tecniche: le immagini acquisite sono state filtrate per rimuovere il rumore e successivamente la porzione dell’immagine relativa alla mano è stata estratta valutando le informazioni sul colore e l’istogramma modello. La posizione della punta delle dita viene calcolata utilizzando le informazioni relative alla curvatura dell’area corrispondente alla mano, e non mediante l’utilizzo di marker sulle dita. Infatti, l’utilizzo di marker colorati richiederebbe all’utente di aggiungere oggetti addizionali alla propria mano: il metodo basato sulla ricerca di punti di massima curvatura non richiede invece l’utilizzo di marker. Dato che questo metodo rileva anche dei “noisy fingertips”, è stato implementato anche un innovativo sistema di filtraggio. Il filtro implementato è basato

Figure 1.2: NDI Polaris Spectra Hybrid.

su due tipologie di filtraggio: filtraggio spaziale e filtraggio temporale; il filtraggio spaziale utilizza i frame destro e sinistro catturati allo stesso istante temporale e, combinando le informazioni, è in grado di rimuovere del rumore residuo; il filtraggio temporale invece utilizza due frame catturati ad istanti temporali successivi per eliminare altri “false detect”. Questa tipologia di interfaccia è molto intuitiva ma presenta problemi di affidabilità. La soluzione proposta dall’autore verrà spiegata e successivamente pregi e difetti e possibili sviluppi futuri verranno illustrati.

In Figura 1.3 è possibile osservare l’interfaccia grafica del programma realizzato. La finestra 3D mostra schematicamente, mediante una sfera bianca, la posizione istantanea del dito di fronte alle fotocamere. In basso, per ciascun frame destro e sinistro, si può notare l’area della mano filtrata dallo sfondo e, cerchiati in rosso, i fingertips. Infine, a destra, è presente un menu di regolazione del programma che permette di variarne i parametri operativi.

La tesi è organizzata in diversi capitoli. Il secondo capitolo costituisce un’introduzione alla tesi. Il terzo capitolo espone invece la struttura del sistema utilizzato nella tesi. Nel quarto capitolo, con il fine di capire correttamente gli argomenti sviluppati nella tesi, vengono mostrati alcuni concetti base: vengono fornite le nozioni base riguardanti l’acquisizione delle immagini; vengono poi illustrati gli algoritmi relativi all’image processing, con particolare riferimento a quelli utilizzati nel programma. Molti di essi sono già stati sviluppati in passato e sono quindi inclusi in una libreria di nome OpenCV; altri algoritmi sono invece stati sviluppati dall'autore e verrannò quindi maggiormente approfonditi.



Threshold	128
Erode	1
Dilate	1
Smoothing	10
K-Curvature	40
Angle	90
Holes	2
Height Range	15
Width Range Negative	10
Width Range Positive	70
Calculation Window Size	1
Search Window Size	1
Color Distance	30
Time Filtering Window Size	30
Time Filtering Minimum Reliability	5

Figure 1.3: Finestre del programma.

Nel quinto capitolo viene riportata ed analizzata la struttura completa del programma; esso è strutturato in più oggetti aventi diversi obiettivi: in questa parte della tesi, gli oggetti vengono presentati in modo tale da far capire al lettore la struttura del programma scritto in C++; la conoscenza della struttura del programma permette infatti di poter introdurre future modifiche. Successivamente, vengono documentati i risultati ottenuti in fase di runtime e vengono discussi pro, contro e possibili sviluppi futuri. Alla fine vi è una conclusione che riassume i punti principali della tesi. Viene inoltre fornita una bibliografia alla fine.

Chapter 2

Introduction

In the past years, many different human-machine interfaces have been developed by engineers, in order to let a computer understand the movements desired by the user. In fact, most of the times, in engineering applications, it is necessary to control the position of a system depending on the user commands. There are many kinds of human-machine intefaces, but all of them can be classified in two big categories [1].

- Human-machine interfaces requiring physical contact. These interfaces are the older ones. When technology started to be developed, interfaces required a physical contact. For example, the user needed to press a button, move a lever, turn a knob and so on. In the last few years, new interfaces started to appear: they were based on joysticks or joypads or similar devices because they're intuitive. Both of these interfaces require the user to touch or press, so they all need a physical contact. In Figure 2.1 an example of physical contact interface is shown: it is a surgery robot created by the Kagawa-Kawashima Lab in Tokyo Institute of Technology.

- Human-machine interfaces not requiring physical contact. They don't need the user to directly interact with an interface, because the interface is not physically touched. For example, speech recognition and computer vision belong to this category. In this case, microphones, cameras and other kinds of sensors are used .

Both of them have pros and cons. Sometimes, when the user needs to control a machine, it could be very useful to do that without any contact with

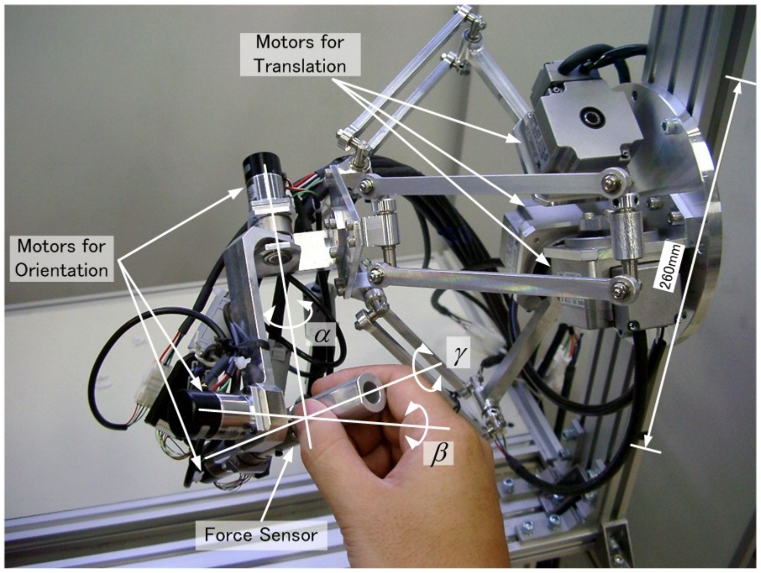


Figure 2.1: Kagawa-Kawashima Lab surgery robot.

the machine. For example, it could be good to command some functions just by using the voice, or by moving a specific part of the body, such as the hand or an arm [2]. On the other hand, these new kind of interfaces sometimes cannot be completely reliable due to many factors. For example, interfaces based on audio signals cannot work if the environment is noisy and also voice signals change depending on the user's voice and depending on the environment characteristics. Video interfaces based on cameras have similar problems: they're sensible to environment characteristics, such as light conditions and object color conditions; also, there could be other problems related to occlusion, for example if an additional object is interposed between the camera and the user.

This means that sometimes it is very difficult to choose between interfaces with or without physical contact. On one hand, the first ones are reliable but they need to be touched, so they need the user to be close to them. On the other hand, the second ones are more intuitive and natural because they don't need the user to touch any apparatus, but they still have many reliability problems. In the future, probably technology will prefer the second ones - interfaces without physical contact - because they're closer to the human behavior. In fact, when humans interact with each other, most of the times informations are given using voice and human movements, without touching the other people. But in order to do that, engineers still have to face many problems regarding reliability and stability; in fact, a new interface can be used in applications only if it satisfies particular reliability conditions. For example, an interface doing a critical work has to be very reliable, or it can cause many damages to people or the loss of money. That's why many engineers and scientists are focusing on these new interfaces and are trying to improve the reliability. Many good results has already been achieved in the game and entertainment area, because in these fields reliability requirements are not so high. But in industrial and special fields, reliability is the most important requirement and that's why this new kind of interfaces still has many problems to be sold in the market. Much research has been done to undestrand the human gesture: the approach can be done by using only one camera, if the purpose is just to understand the gesture [3] and it is not needed to calculate the position of the hand in the three dimensions [4]. Understanding the gesture can be useful because a gesture can be associated to a particular order given by a human [5]. One of the most useful parts of the human body, to give commands, is the finger: many researches have been focused in finding the position of a fingertip in an image, also combining many algorithms to improve the reliability of the method [6]. Also, some studies have been done to isolate the human skin areas is an image by evaluating the color [7]. Some other researches are focused on the stereo vision: by using two cameras, it is possible to calculate the distance of an object in a three dimensional space [8] or making a 3D object reconstruction [9]. The position of a fingertip can be calculated by using a hand 3D model [10], but also another technique is to use the hand area curvature information [11]. In this thesis, the hand curvature information has been used; in addition to that, some innovative filters have been implemented to filter the noisy fingertip candidates.

The aim of the author is to show an innovative interface used to track the position of a fingertip in a three dimensional space. The recognition of the fingertip is done using "stereo vision", i.e. using two cameras. The program, for example, is able to track the position of the index fingertip moving in the space. This application could be useful in many fields that require haptic interfaces. The program gets the position of a fingertip point (in three dimensions coordinates). This position can be sent to an actuator which reproduces the same position but in another location. A simple example could be the following: the user wants to move a heavy object in a manufacturing plant. This can be done by using a physical contact interface, such as a

	Kinect	NDI Polaris	Stereo vision
Markers	no markers	passive/active	no markers
Application	entertainment	medical	entertainment, industrial
Resolution	low (3 mm)	high (0.5 mm)	low (2 mm)
Cost	low	high	very low
Reliability	high	high	must be improved

Table 2.1: Comparison between different kinds of human-machine interfaces.

joystick with some buttons; but it is also possible to do that retrieving the position of the user fingertip, so that the user just have to move his finger in the free air, and the computer can understand its position in the three axis x, y, z. When this position is retrieved, it is sent to an actuator using a protocol of communication, and then it can be easily actuated. At the moment, on the market, other interfaces can meet these requirements. In Table 2.1 a comparison has been made between different tracking technologies. Kinect technology is used in the Microsoft X-Box system: it integrates an RGB camera sensor and an infrared distance sensor; its distance resolution is not high (3 mm) but the cost is low because this technology is used in the game industry. NDI Polaris (Figure 2.2) is a position tracking system used in the medical field to track the position of some markers; its resolution is high but also the cost is high and it requires physical markers. The proposed stereo vision technology can be applied in the fields that require a low cost: in fact, the system is very cheap because it does not require an additional distance sensor. Possible targets are the entertainment field and, according to some reliability improvements, also the industrial field. The spacial resolution of the system depends on the resolution of the cameras.

This project has been developed combining many techniques: the acquired images are filtered to remove the noise, and then the hand area in the image is extracted by evaluating the color of the pixels and the color histogram. The position of the fingertip is calculated by using the hand area curvature information and not by using color markers on the fingertips. In fact, the use of color markers requires the user to add additional objects to his hand; the hand curvature method does not require the user to wear additional markers. Since this method produces noisy fingertips candidates, an innovative method to filter the noisy points has been implemented. The filtering method is based on both spacial filtering and time filtering: spacial

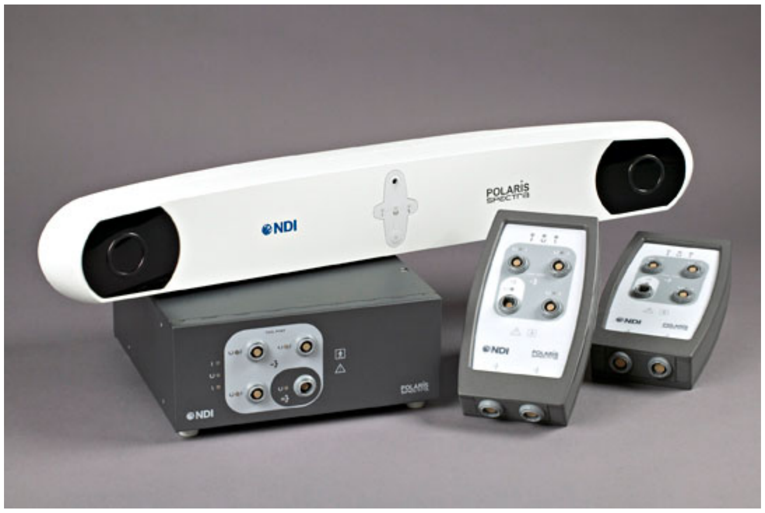


Figure 2.2: NDI Polaris Spectra Hybrid system.

filtering uses the left and right frames grabbed at the same time; on the other hand, the time filtering uses two frames grabbed at different times to delete the noisy fingertips. This kind of interface is very intuitive but it still has problems concerning reliability. The author solution will be explained and then pros, cons and future improvements will be explained.

The thesis is organised in different chapters. In the beginning, in order to easily understand the thesis, the experimental setup and many preliminary and basic concepts are explained: simple notions about image acquisition are shown. After that, the image processing algorithms are explained, with particular focus on the algorithms implemented in the program. Many of them have already been programmed before, so they're included in the OpenCV image processing library; some other algorithms have been developed by the author, and so they're deeply explained. Then, the complete structure of the program is analysed and reported; the program is structured into different objects which have different targets: in this part of the thesis, all these objects are explained, in order to let the reader undestand the C++ program code; so, the program can be easily modified and some parts can be copied into a new program. Moreover, some results are reported, and pros and cons are discussed. At the end, there is a conclusion resuming the main points of the thesis. A bibliography with references is provided at the end of this book.

Chapter 3

Stereo vision experimental setup

In this chapter, the experimental setup is described. Basically, the system is divided in two parts: hardware and a software. Two cameras have been used to implement stereo vision; the cameras are connected to a personal computer which runs a software created by the author. Both hardware and software will be analysed.

3.1 Stereo vision system

The system is composed by two high speed IEEE 1394b Firewire connection cameras (PointGrey FLEA2 cameras) connected to a personal computer running Ubuntu Linux. Two cameras are necessary to implement an object 3D position retrieving. The operation of stereo vision is similar to the human view; in human beings and many animals, the informations retrieved by two eyes are used by the brain to understand the spacial position of the objects in the environment. In order to calculate the position of an object, it is necessary to identify the position of the same object in two images; after that, by matching the informations in both left and right images, the 3D position is computed. In Figure 3.1 the experimental setup is shown.

The two cameras are mounted on a tripod, and the the distance of the camera sensors is 110 mm. The camera specifications are the following: the sensor is a Sony CCD sensor with a maximum resolution of 648x488 pixels. The integrated ADC has a 12 bit resolution and camera provides an internal RGB conversion. The maximum framerate at 640x480 is 80 fps depending on the maximum communication speed of the bus; transfer rate goes from

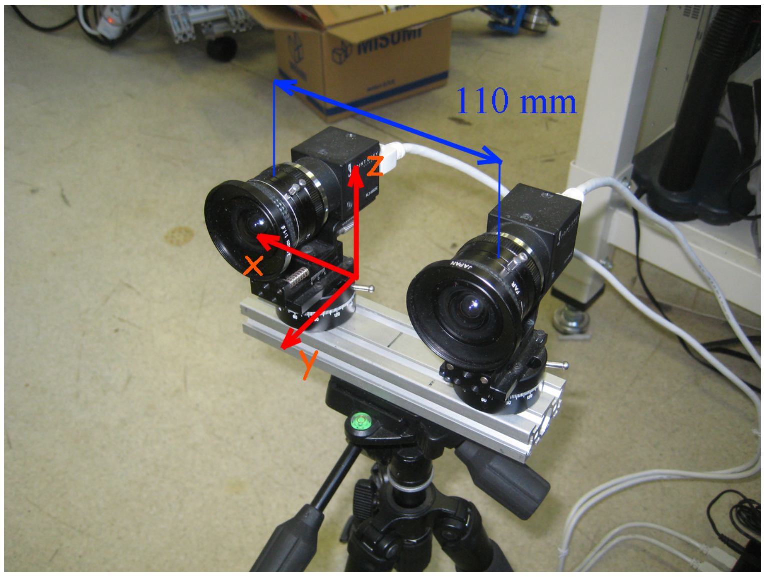


Figure 3.1: Stereo vision system.

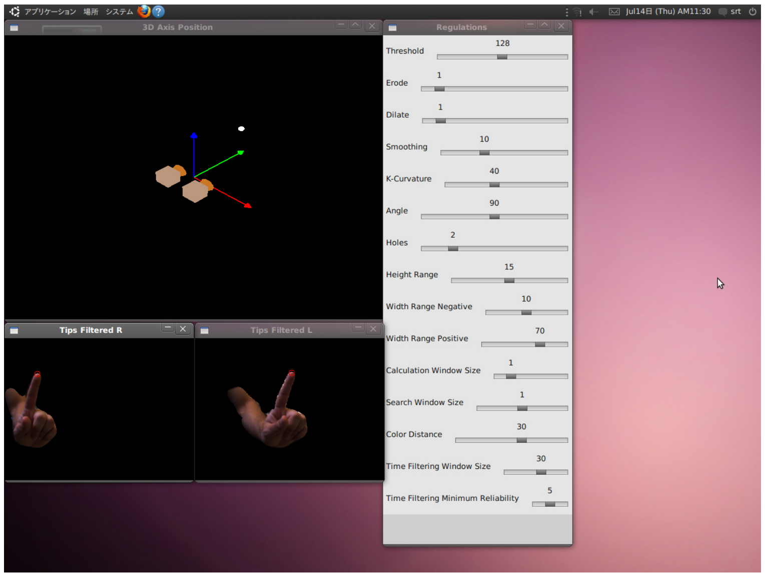
100 Mb/s to 800 Mb/s. In this thesis, a camera resolution of 640x480 has been used; the two cameras are sharing the same IEEE 1394 bus with a 400 Mb/s speed; the camera grabs frames at 15 fps (frames per second) - i.e. a new frame is grabbed every 66 ms.

The cameras are provided with Spacecom lenses with a focal length of 3.5 mm. The lens distortion is assumed to be small, so no additional distortion corrections have been implemented on both hardware and software sides.

3.2 Software

The two cameras are connected to a personal computer running Ubuntu. Ubuntu is a very famous open source operative system and it's good for programming. Many libraries about graphics and image processing have been developed for Ubuntu Linux. Most of the libraries are open source libraries, so software developers don't need to pay to use them. Thereafter, the cost of the system is only given by the two cameras and the personal computer running the main program.

The fingertip tracking program has been developed using C++ language plus additional open source libraries. C++ language is an object programming language: it combines high level programming approach with good performances. Many features directly derive from C language programming; in addition to that, C++ lets us divide variables and functions into classes, so that the program can be more easily undestood and modified. Other additional libraries has been used: "libdc1394", dedicated to IEEE 1394b cameras image acquisition, OpenCV library [20, 21], dedicated to image processing, and OpenGL library, used to create the 3D graphic interface [22, 23][22, 23]. The program graphic user interface can be seen in Figure 3.2.



Threshold 128
Erode 1
Dilate 1
Smoothing 10
K-Curvature 40
Angle 90
Holes 2
Height Range 15
Width Range Negative 10
Width Range Positive 70
Calculation Window Size 1
Search Window Size 1
Color Distance 30
Time Filtering Window Size 30
Time Filtering Minimum Reliability 5

Figure 3.2: Program windows.

First of all, the program is trained with hand color informations. Then, a main cycle is repeated until the user ends the program. Using the hand model color informations, the program identifies the hand region in the left and right camera frames. Evaluating these informations, the fingertips positions are retrieved and filtered to remove the noise. Then, the most reliable finger is taken as the index finger, assuming that the user is using only the index finger to indicate the position. To ensure this, many image processing algorithms has been implemented [14]: some of them has already been used in other projects and others have been proposed by the author and will be discussed in this thesis.

The tracked fingertip is then shown in a 3D window which is shown in Figure 3.2. Also, it is possible to change the parameter values in real time;

in fact, the user can change the program parameters during the execution of the program, so it is possible to understand the influence of each parameter in realtime. When the program starts, the default parameters are loaded; these parameters have been empirically estimated by the author and they can be considered good in an average situation.

Chapter 4

Image acquisition and processing

Before analysing the project, it's important to understand the basic concepts and algorithms implemented in the software. Many image processing algorithms have been used: some of them are well known algorithms already implemented in many other softwares. Other algorithms have been created ad hoc by the author. This chapter is divided in two sections: the first one is mainly about camera physics and image data acquisition using camera sensors; the second part explains the algorithms so it is mainly focused on the image processing and feature extraction. The concepts explained in this chapter are preliminary to the understanding of the software.

4.1 Image data acquisition

The purpose of this section is to let the reader understand the basic notions about how images are acquired by a digital system [16]. In order to apply useful algorithms, it is necessary to transform analog informations to digital informations that can be more easily computated by a digital computer. The interface that does this task is the camera sensor. The purpose of a digital camera is to receive the light from the environment, process it in the analog domain, convert it in a digital domain and send it to a computer. This goal is obtained following many steps, and each part of the camera has a specific intent. To understand this part of the system, it is necessary to have basic knowledge of both physics, electronics, hardware and software. In fact, the acquisition of an image is merely a physical and electronic process; but the representation of informations requires also informatics and mathematics knowledge.

4.1.1 Camera image acquisition

Basically, a digital camera is a light sensor. A camera receives the lights from the environment and transforms the light information into another kind of information; a device doing this task is called "transducer". In particular, in a digital camera, the input information is the light from the environment. The light information is converted in an electrical information, because it can be easily manipulated. For example, in a CCD (Charged Couple Device) camera sensor, the light information is converted in an electrical charge information. A digital camera is composed by different parts:

- Optical element. The optical element basically consists in one or more lenses. Their function is to concentrate the light rays of the environment to the camera sensor. The quality of the captured image is mainly affected by the optical element. Different optical elements can direct the rays to the sensor in different ways: by consequence, the image obtained is different. In particular, varying the optical element characteristics, it is possible to change the angle of view, i.e. the zooming factor of the camera. Cheap or bad optical elements cause many problems to image acquisition because they introduce distortions and aberrations that sometimes can be hardly corrected by software.

- Camera sensor. A camera sensor is a matrix of light sensors. In fact, when the camera grabs a frame, the result image is a matrix of numbers; this is because the camera does a spatial discretization. A continuous distribution of lighting values is transformed into a discrete distribution of lighting values. Each value is captured by a small sensor in the matrix camera sensor. This small sensor is called "photosite". Normally, cameras has several thousands of photosites: the total number of photosites is called "resolution". If resolution is higher, the discretization introduces less approximations, but also we need to analyse a bigger amount of data. The main purpose of the sensor is to receive the light concentrated by the optical element and transform each light information of the photosites in an electrical information. In each photosite, only a portion of the light spectrum is filtered and evaluated; photosites can be sensible to three kind of lights: red, green or blue (RGB convention). This particular structure is called "Bayer pattern" and is shown in Figure 4.1. The electrical information can be a charge or a voltage. The charging (voltage) value is proportional to the energy of light received by the camera sensor, so it's obtained by physical integration of the light power input. This means that to get a correct exposure, it is necessary for the sensor to receive the correct amount of light energy, so it will take some time to grab a frame; this time is called "exposure time" and it depends on the light conditions of the environment and on the optical element characteristics.

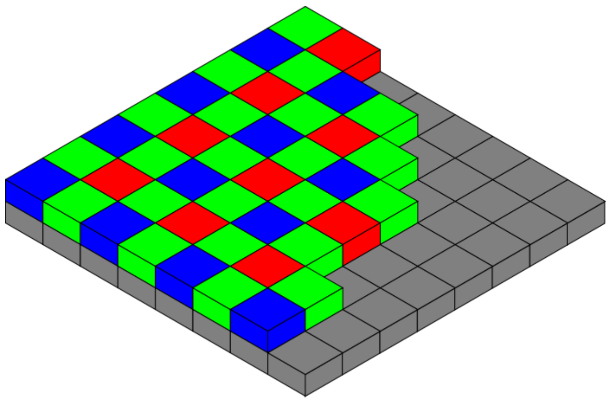


Figure 4.1: Bayer filter on camera sensor.

- Additional electronics circuits. The purposes of the additional circuitry are many. First of all, it is necessary to convert the analog voltage values obtained from the sensor matrix to digital values. In fact, digital computers work with digital information, so it is necessary to transform an analog information into a digital information that can be easily computated and manipulated. Before converting the analog voltage values to digital numbers, it is possible to amplify the signal from the sensors; this process might be very useful in low light environments, because the low analog value (due to low light) can be amplified; however, the amplification process introduces noisy digitalised images that need to be filtered using a software filter on the personal computer side. As soon as the analog informations are converted to digital, using an ADC (Analog to Digital Converter), they've been discretized. Each sampled value in the matrix has a specific resolution, given by the number of bits for each photosite. For example, a 8 bit resolution can represent a total of 256 values from 0 to 255. Normally, the ADC resolution is more than 8 bits, for example 10, 12 or 16 bits. However, when the image if compressed and analysed by the software, normally it is scaled to a 8 bit resolution. After the A/D conversion has been done, additional hardware circuits provide the data transmission from the camera to the personal computer. The transmission can be done using different communication interfaces and protocols: for example, Universal Serial Bus (USB) and Firewire bus (IEEE 1394).

To realize the project, a Firewire IEEE 1394b camera model has been used. Stereo vision has been implemented using two PointGrey FLEA2 cameras. The specifications are the following. The spatial resolution is 640x480 pixel (0.3 megapixel). The ADC has a 12 bit resolution. The data transfer rate of the Firewire bus is up to 800 Mbit/s. Each camera can grab frames at a 80 fps framerate. However, the maximum grabbing framerate is limited by the maximum bus bandwith (800 Mbit/s).

4.1.2 RGB and HSV color spaces

The purpose of a digital camera is to convert the light analog information into a discrete digital information (a sequence of numbers). For each pixel in the image matrix, the color information can be coded in different ways depending on the convention. In Figure 4.1 the structure of a sensor is shown. There are three different colors: red (R), green (G) and blue (B); this is the so called RGB convention. In fact, it is possible to approximate the color information of a pixel using only three values; they can express the light exposure spectrum of each pixel. Red, green and blue conventional colors have a specific spectrum power distribution. Combining RGB functions together, multiplying each other by a weighting coefficient, it is possible to approximate every visible color. In fact, the human visible spectrum is restricted to a specified wavelenght band from 380 nm to 680 nm. In Figure 4.2 the RGB conventional color functions are shown.

For each photosite, the camera retrieves only one value (R, G or B); the other two values are calculated using a demosaicing algorithm, evaluating the color value of the neighbor photosites. At the end of this calculation process, each pixel has three color value (RGB information). Each value has a specific depth in terms of resolution, for example 8 bit is the most common. It means

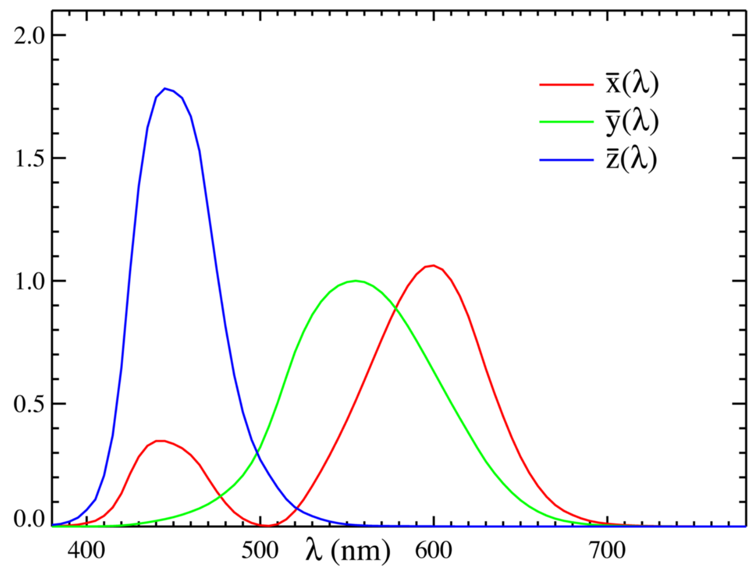


Figure 4.2: RGB color functions.

that each pixel has 24 bit information (3 bytes - 1 byte equals to 8 bits). On the physical sensor, usually the photosite composition is 50% green, 25% red and 25% blue: this is done because human eye better distinguishes colors in the green band; during the acquisition process, it is better to sample the green color with a enhanced spacial accuracy, in order to let the demosaicing algorithm estimate the real color with a higher accuracy. With 8 bits per channel (a total of 24 bits per pixel) it is possible to codify 2^{24} different colors, more than 16 millions.

Important notions are brightness and saturation. The brightness of each pixel is determined by the RGB values; more specifically, brightness is proportional to the mean value of RGB. The brightness is a number indicating the light power of each pixel; if this value is high, the light is perceived as strong light. Pixels with (255, 0, 0) and (0, 255, 0) - respectively pure red and green - have the same brightness but the color is completely different. In fact, the kind of coded color has to be expressed also using other two parameters: hue and saturation. Hue indicates the tonality of the color, for example violet or brown, while saturation indicates the purity of the color. The values of brightness, hue and saturation of each pixel can be calculated using the RGB values. There is a bijective correspondance between RGB space and a brightness, hue and saturation space.

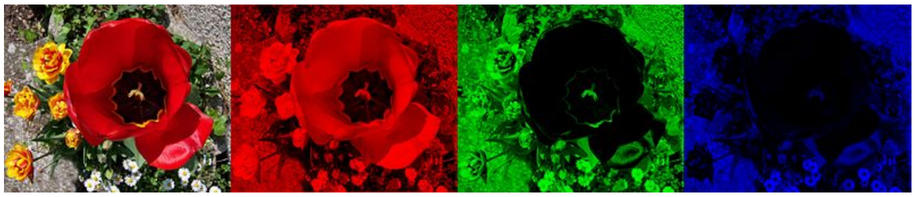


Figure 4.3: RGB image decomposition.

In Figure 4.3 it is possible to see how an image can be decomposed into the RGB data information planes. The image contains pure colors (red and green) but also grey. Greyscale colors contain both red green blue at the same quantity; it means that the pixel colors (0, 0, 0), (128, 128, 128) and (255, 255, 255) are all greyscale colors; the only difference is given by their brightness. In fact, 0 value is black, 128 is a normal grey while 255 is white. This is to introduce the notion of saturation: if the values of RGB are very close, the pixel color is not much saturated, so the saturation value is low; but if the RGB values are very different, the saturation is high and the color is perceived as "pure". For example, (128, 128, 128) is a zero saturation point (grey) and (0, 0, 255) is a high saturated blue (the purest blue).

By the way, only by evaluating the numeric values of a RGB point, it is very difficult to understand which color has been coded. So, a more intuitive convention is used in specific applications; it is known as HSV (Hue, Saturation and Value). Instead of coding each pixel with RGB information, it is possible to encode the light information using HSV numbers. Similarly as RGB, the HSV convention uses a bit depth for each information; for example, 8 bits for Hue, 8 bits for Saturation and 8 bits for Value, with a total of 24 bits. Each pixel information can be easily converted from RGB to HSV using specific formulas. The convertion total calculation time is low but it gets higher as the image resolution gets higher. For example, if the calculation time for a 320x240 pixel image is T, the calculation time for a 640x480 pixel resolution image is 4T; in fact, if the image height and width are doubled, the total number of pixel in the image is four times higher and so also the calculation time is four times higher.

In Figure 4.4, a three dimensional HSV cylinder is shown. The V value is related to the brightness, while H is related to the color hue and S is related to the color saturation. For each RGB pixel, the correspondant HSV values can be calculated using easy formulas. Let us define M, m and C: these are

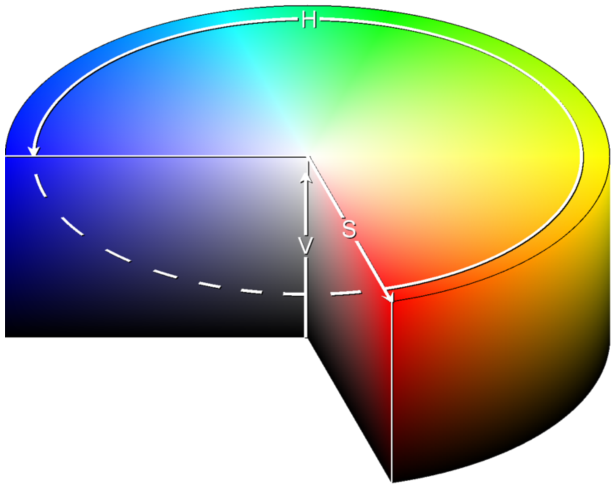


Figure 4.4: HSV cylinder.

respectively the maximum and minimum values of RGB, and the chroma value. The calculation method of H, S and V is described above.

$$V = M = max(R, G, B)$$

$$m = min(R, G, B)$$

$$C = M - m$$

$$H' = \begin{cases} undefined & , C = 0 \\ \frac{G-B}{C} \, mod \, 6 & , M = R \\ \frac{B-R}{C} + 2 & , M = G \\ \frac{R-G}{C} + 4 & , M = B \end{cases}$$

$$H = 60^\circ \cdot H'$$

$$S = \begin{cases} 0 & , C = 0 \\ \frac{C}{V} & , otherwise \end{cases}$$

The representation in the HSV color space is very useful in the analysed project because the color information is virtually independent from the light intensity of the environment. In fact, using the HSV color space, the object can be identified creating a statistical model evaluating only the Hue and Saturation color planes. The mentioned color statistical model of an object is called histogram and will be discussed in the next sections of this chapter. The program converts the RGB image information to the HSV color space and then filters the hand area using a color model of the hand; the color model is a bidimensional probability histogram using both the Hue and Saturation planes; however, the Value plane is not used.

4.1.3 Exposure, saturation and white balance

The light reflected by the objects in the environment reaches the camera sensor and determines the exposure. In other words, the energy flowing with the lights is converted to electrical energy that charges the sensor photosites. This process is not instantaneous but it requires some time to be performed. In fact, the light rays reaching the sensors have specific power that depends on the environmental characteristics. For example during the daytime the light is stronger than at night; by consequence, when light is weak, it is necessary more time for the sensor to charge correctly. This charging time is called exposure time. If the light in the environment is low, it is possible to amplify the sensor signals interposing a amplifier stage between the sensor and the ADC; however, icreasing the gain also increases the electronic noise produced by the sensor. As a consequence, the image is noisy and sometimes it's hardly possible to filter it; this phenomenon is very common when taking pictures at night time, in low light conditions: in order to limit the exposure time, the camera needs to increase the analog signal gain; the experienced result is a noisy image that needs to be filtered.

In many cameras, the exposure parameters cannot be set manually by the user because they're automatically calculated in real time by the camera. The esposure time and gain are set depending on the light conditions of the environment, and the target is to obtain a well exposed image. This means that the mean color value of the image has to be around a specific value. For example, in the RGB values are coded using 8 bits, the maximum value for each color is 255; so, the brightness (mean color value) is in the range between 0 and 255. It is possible to regulate the exposure parameters so that the mean color value is around 128, approximately half of the range. Of course, 128 is just an example value and the exposure control can be set in a different way. However, this automatic regulation is not good in image processing: in fact, after regulating the exposure parameters at the start of the program, it is better to mantain them constant. This choice is necessary because the program uses a color histogram model of the hand. If the exposure parameters change in the camera due to the camera automatic control, it is possible that the hand acquired color changes depending on the exposure parameters change; by consequence, hand pixels are not recognised and the next steps of the algorithm are not functioning well. Because of this reasons, the esposure parameters - i. e. exposure time and gain - are manually set by the user at the start of the program and after that they're mantained constant. They can be manually modified during the program runtime, but it's reccomended to create new hand color histograms after the exposure parameters editing is performed.

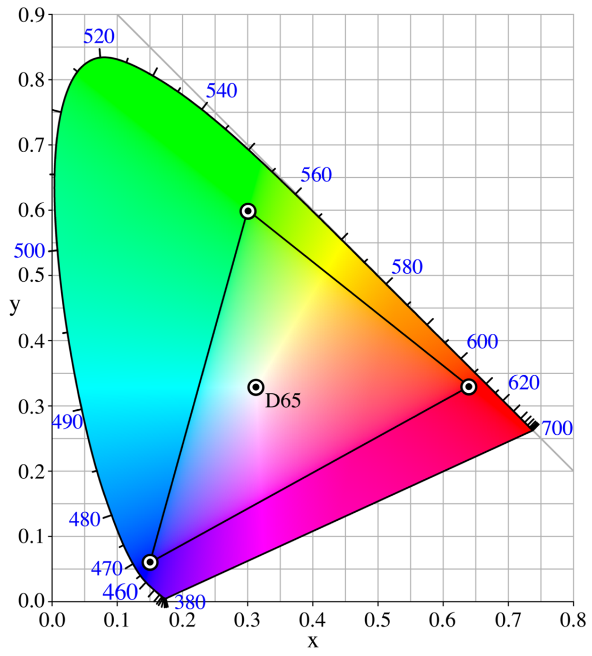


Figure 4.5: Gamut.

The automatic camera control also decides two other features: saturation and white balance. This features are critical because they also change the color hue and saturation of each pixel. The notion of saturation has already been explained in an intuitive way. In Figure 4.5 a rigorous explaination image is shown. The figure is called "gamut", and basically it is a map of all the colors that can be obtained mantaining the brightness constant. The values of x, y and z are respectively the values of red, green and blue. As the brightness is constant, the sum of x, y and z is equal to one; the three values are not independant, and changing two of them, the third can be determined. The center of the gamut is the greyscale white point (color is not saturated), and moving toward the vertexes the saturation increases - the three vertexes are red, green and blue. Normally cameras, after acquiring the digital image, automatically correct the image saturation of each point according to a control function; this automatic feature has to be disabled because it affects the color information at runtime.

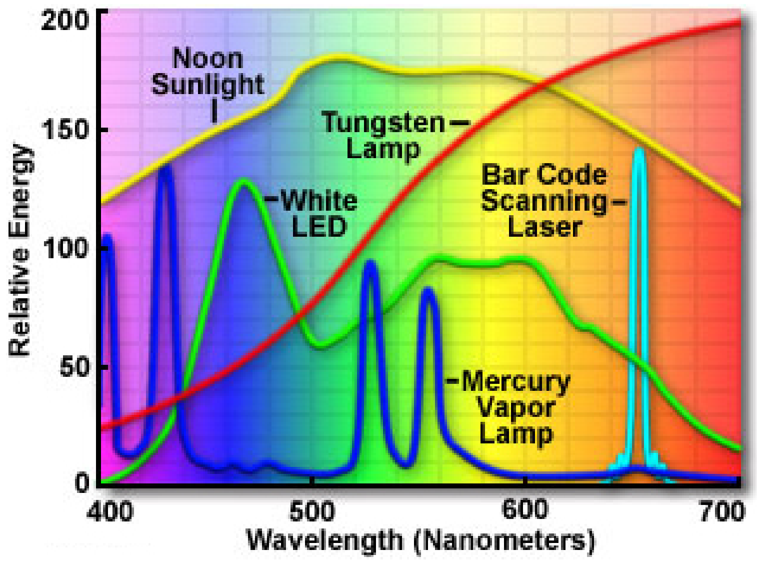


Figure 4.6: Lamp spectrum.

Another interesting feature is the white balance feature of the camera. It is used to correct the effect of the light sources spectrum. It is well known that the light that reaches the camera sensor is the light reflected by the objects in the environment. Each object reflects the light received by one or more light sources. Examples of light sources are sunlight and lamps. The light spectrum is different depending on the kind of light source. For example, different kinds of lamps have different light spectra; in Figure 4.6 different lamp spectra are shown. The purpose of white balance is to correct the lamp spectrum effects, and it is mainly used in photography, but it is not necessary in this image processing project. In fact, automatic saturation and white balance correction controls imply that the correction parameters are time variant and they can create problems to the program operation. So, automatic control has been disabled and the user can change the parameters manually at runtime. As in the case of exposure, it is reccomended to update the color histogram model of the hand after a manual parameter editing is performed.

4.1.4 Image noise filtering

During an image acquisition, the sensor adds electronic noise to the image data. This noise is present expecially when using high analog gain values: in fact, the photosite signals from the sensor are amplified, but also the noise is amplified. As a consequence, it is necessary to filter the image before applying image processing algorithms evaluating the color. Basically there are two kinds of noise:

- additive noise, when each pixel color differs from the real color just a little. Tipically, it is distributed in all the image.
- impulsive noise (salt and pepper), when the difference between the pixel color and the real color is higher. Tipically, it is distributed just in a small part of the image.

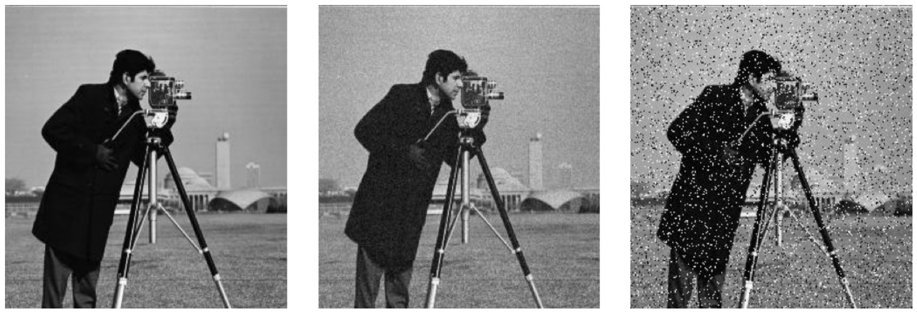


Figure 4.7: Image noises.

In Figure 4.7 different kinds of noise are shown. In the left image, no noise has been added; in the middle, an image with additive noise is shown (gaussian noise); on the right side, an impulsive noise (salt and pepper) has been added. To remove the noise, a smoothing filter has to be appplied to the image. Smoothing the image with a low pass filter is very usesul because the filter removes the noise, but oversmoothing can also cause problems; when an oversmoothing is applied, the image has blurring and the details are lost. For example, mean filter and gaussian filter can be used. Each filter is applied by calculating a convolution between the image and the filter window. When applying a mean filter, the new value of each pixel is simply the mean value of the pixels in a specified window. Below, it is reported a simple 3x3 mean filter window: the new value of each pixel is obtained by calculating the mean value of a 3x3 window containing 9 pixels.

$$W_{mean} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

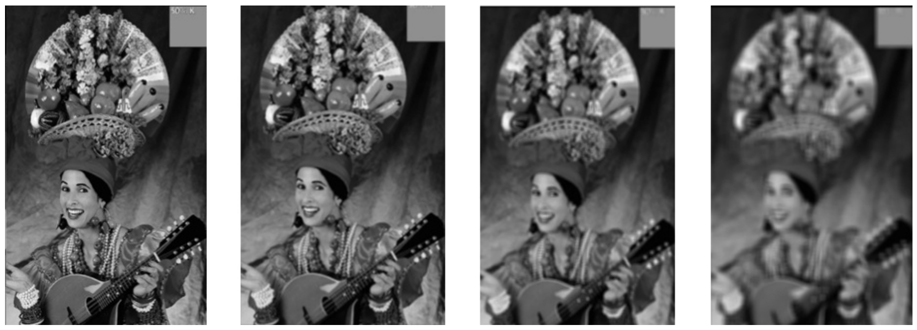


Figure 4.8: Mean filters.

The effectiveness of a filter changes depending on the filter window dimension. By increasing the window dimensions, the smoothing effect becomes higher but also a blurring effect is introduced. In Figure 4.8 the effects of multiple mean filters have been reported; from left to right you can see the original image and three mean filters with different window sizes, respectively 3x3, 5x5 and 9x9. As the window dimension becomes higher also the calculation time grows. The mean filter is very easy to be implemented; another example of low pass filter is the gaussian filter. In the gaussian filter, pixels have decreasing weights depending on their distance from the center. An example of a gaussian filter window is shown below.

$$W_{gaussian} = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

The gaussian filter gives a higher weight to the pixels in the middle so the blurring effect is lower. Mean filters and gaussian filters are useful to remove an additive noise, for example electronic gaussian noise. To remove impulsive noise, an other kind of filters is more effective: the median filters; however they won't be analysed in this thesis. In the project, the acquisition resolution for each camera frame is 640x480 pixels. Then, the image is scaled to half width and weight and it becomes 320x240 pixels. This operation is necessary in order to decrease the calculation time of the following image processing algorithms. After that, a simple 3x3 mean filter is applied to each frame. A 3x3 window is a good compromise because the noise is successfully filtered and the blurring effect is mantained low. Figure 4.9 shows the result of the mean filtering described above; on the left and right sides there are, respectively, the original and filtered images.

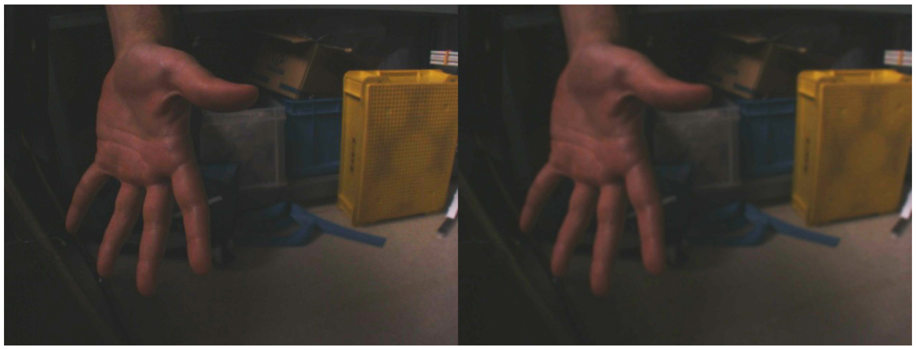


Figure 4.9: Example of mean filter.

4.2 Image processing algorithms

In the first section of this chapter, basic notions related to image data acquisition has been explained. A proper image acquisition is important because additional image processing algorithms are performed using the frames grabbed by the camera. So, it is necessary to obtain good quality image frames; more specifically, the exposure has to be correct and also the noise has to be minimised. Assuming that the image acquision step has been performed correctly, additional image processing algorithms evaluate the frames to extract useful informations. In this section, the image processing algorithms will be analysed; some of them are well known algorithms already implemented in

OpenCV libraries: they can be easily used because the functions has already been written. However, some algorithms have been implemented by the author because such functions were not present in the mentioned library. For example, a contour filter has been implemented in order to smooth the hand contour; also, the fingertip matching algorithm and filtering proposed by the author are innovative algorithms because they are using more grabbed frames to remove the noisy data.

4.2.1 Color probability histograms

One of the most useful instruments is the color histogram. A color probability histogram is a N-dimensional probability distribution, where N is the number of color space planes used. Color histograms can be built using both RGB or HSV color spaces. Basically, they contain informations about the color in a specified window: a color histogram can be created evaluating the entire image but also evaluating only a specific ROI (Region Of Interest). In Figure 4.10 RGB histograms are shown. For each color plane (Red, Green, Blue), a statistical distribution is calculated using the pixels in the whole image. Let us suppose that the RGB image has a color depth resolution of 8 bits per channel; this means that in each pixel, the values of RGB are in the range between 0 and 255. In this case, the histogram has the numbers from 0 to 255 on the abscissa axis. For each number in the range, the histogram value (ordinate axis) is the total number of pixel having the same RGB number. For example, if the value of the red histogram is M in the B bin, it means that in the image, M pixels have the same B red value.

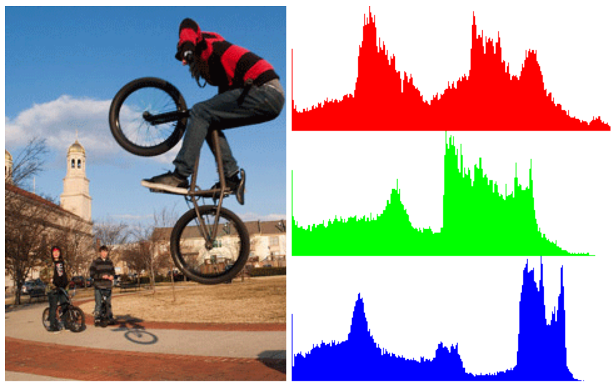


Figure 4.10: RGB Histograms.

In Figure 4.10, all the points in the image have been evaluated to calculate the histogram. By the way, it is also possible to select a region of interest. In our case, a color histogram is created using hand images; it means that a color histogram model is created using the hand data. After the program is trained with a histogram, it is possible to use the histogram to evaluate the hand pixels in the next frames. Color histograms can be calculated using all the color informations of the RGB planes, but also using only one or two specific planes. A histogram can be built also by using the HSV color space information. In this project, the HSV space has been used but only the Hue and Saturation planes has been evaluated. In the previously mentioned figure, three 1-dimensional histogram have been calculated, one for each color (red, green and blue). However, it is also possible to calculate only one 3-dimensional histogram. In a N-dimensional histogram, the histogram space dimension is N; in fact, N dimensions correpond to the color planes, and each N-tuple has a particular value proportional to the number of pixel in the ROI that have the same planes values. For example, if we build a 3-dimensional histogram in the RGB space, the total number of dimensions in the histogram will be three. If the value of the histogram is 1234 at the histogram coordinates (10, 20, 30), it means that in the region of interest, 1234 pixel have the same RGB values (R=10, G=20, B=30).

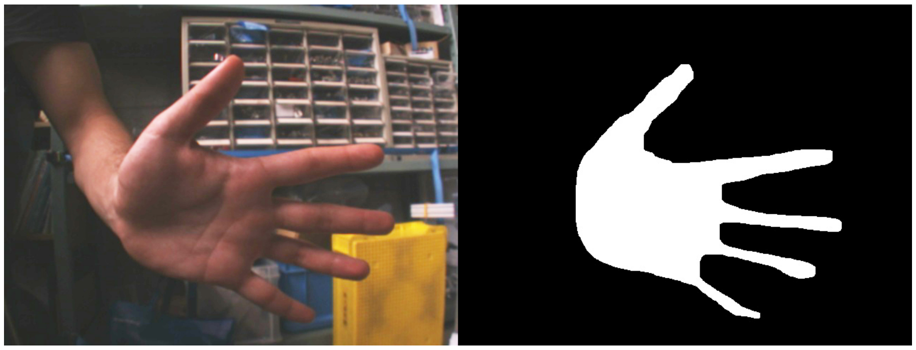


Figure 4.11: Histogram reference images.

In Figure 4.11 it is shown an example of a captured frame, on the left, and the white hand ROI, on the right. Using the ROI pixels color values, it is possible to calculate the histogram. The histogram is discretized using 16 values for each color plane (Hue and Saturation); it means that the obtained 2-dimensional HS histogram has 16^2 values, so 256 discrete positions.

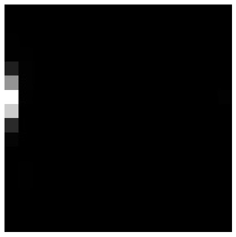


Figure 4.12: Example of 2-dimensional Hue Saturation histogram.

In Figure 4.12 the resulting histogram has been plotted. The two axis represent the Hue and Saturation discretized values. For each bin, the greyscale color is proportional to the number of pixel in the same range. The histogram area with the maximum amount of pixel is the white area, while black areas indicate that no pixels have that particular Hue and Saturation values. The histogram can be read also by a statistical point of view. In fact, a pixel randomly taken from the ROI has a high probability to have hue and saturation values in the white area of the histogram, and low probability to have the same values as black area pixels.

4.2.2 Back-projection

The creation of a histogram is necessary because the back-projection algorithm uses the histogram to calculate, for each pixel in a frame, a probability to be an object pixel. This step is easy to be performed. The back-projection algorithm simply evaluates the color properties of each pixel in the acquired frame and associates a probability value to that pixel. The output image is a greyscale single channel image; for each pixel in the output image, the probability goes from 0 to the maximum value; for example, if the greyscale image has a bit resolution of 8 bits, the probability value is in the range between 0 and 255. The probability value of each pixel is calculated simply by using the histogram as a lookup table.

After calculating the back-projection image, an output grayscale image is obtained. In Figure 4.13 an example of back-projection is shown. The back-projection image is calculated using the filtered frame informations; in fact, the frame captured by the camera is previously filtered using a mean filter with a 3x3 pixel window (Figure 4.9). In the figure, it's easy to see that the hand points are detected as their greyscale color is close to white. But there is also much noise that needs to be filtered: the filtering methods will

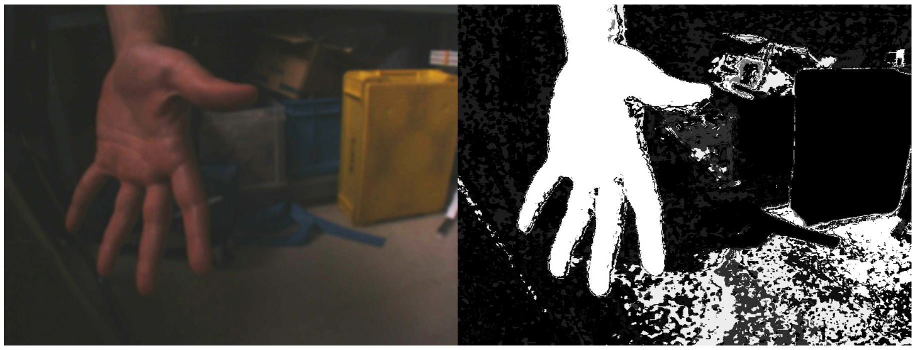


Figure 4.13: Back-projection image.

be described in the following subsections.

4.2.3 Image thresholding

After the back-projection image is calculated, the color is coded in greyscale; this means that each pixel of the image is described by a discrete value in the range between 0 and $2^N - 1$, where N is the number of bits for each pixel (for a 8 bit depths, 256 values can be codified). The value owned by each pixel is proportional to its probability to be a reference object pixel; in fact, this value has been previously calculated by the back-projection calculation algorithm. However, it is necessary to discriminate object pixels and non-object pixels; so, a binary representation (0 or 1) is needed. In order to do that, the back-projection image has to be thresholded. More specifically, each pixel value below the threshold is considered 0 (zero value) and each pixel value above the threshold is considered 1 (maximum value - for example 255). This image processing algorithm is very easy to be implemented; at the end of the process, a binary image is obtained.

As you can see in Figure 4.14 the image obtained by the back-projection process has been thresholded with a threshold of 128; in fact, 128 is half scale (0 to 255); everything below is considered 0, everything above is considered 255. As you can see, some pixels belonging to the back-projection with grey color (not 0 and not 255) have been thresholded successfully, so they have been converted to 0 (black) or white (255).

The back-projection image does not have many different values, so there is not much visible difference between the back-projection and the thresholded images. By the way, the conceptual difference is high: the first one is 8 bit

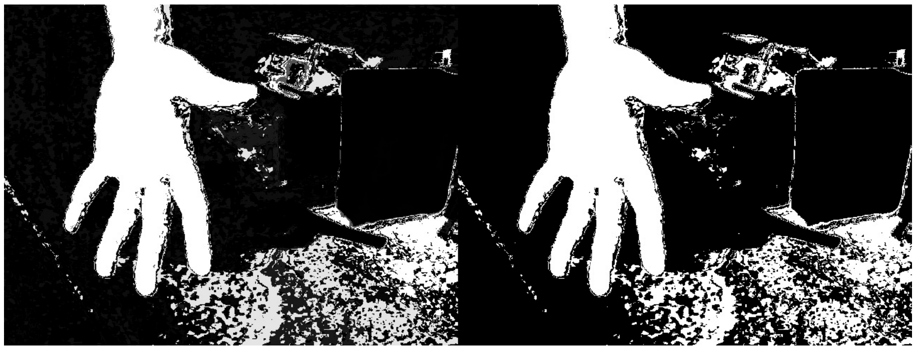


Figure 4.14: Thresholded image.

image, the second one is a 1 bit (binary) coded image. This means that in the thresholded image it is possible to distinguish exactly object pixels and non-object pixels. This clear division is necessary because the next image processing algorithms need binary images to be applied.

4.2.4 Erode and dilate filters

Erode and dilate filters are applied to a binary image. As already described, binary images are images in which every pixel can be represented only by two different values: a zero value or a maximum value; these images are very useful to discriminate pixel appartaining to a specific object and pixel that does not appartain to that; for example, to pixel appartaining to an object, the maximum value is given (255 value, the logic 1) and to pixel not appartaining, the minimum value is given (0 value).

Usually binary images are obtained by analysing the color of a retrieved frame and applying some algorithms; this is the case of this thesis, in which color algorithms and thresholding have been applied. However, many times the mentioned algorithms produce small noisy areas (for example, each noisy area can vary from 1 pixel to 10 pixels); it means that for some points, the binary positive value is given even if the point does not belong to the reference object. This kind of noise can be determined by different causes:

- the first case is when noisy pixels have the same color of the reference object in the reality. In this case, usually noisy areas are big and cannot be filtered effectively. In fact, color based algorithms only function if the color of the target object is different from other objects in the background.

- the second case is when noisy pixels have the same color in the grabbed frame, but not in the reality. This happens when camera adds noise to pixels and so the real color of some pixels is changed. The erode algorithm can be very useful in this case.

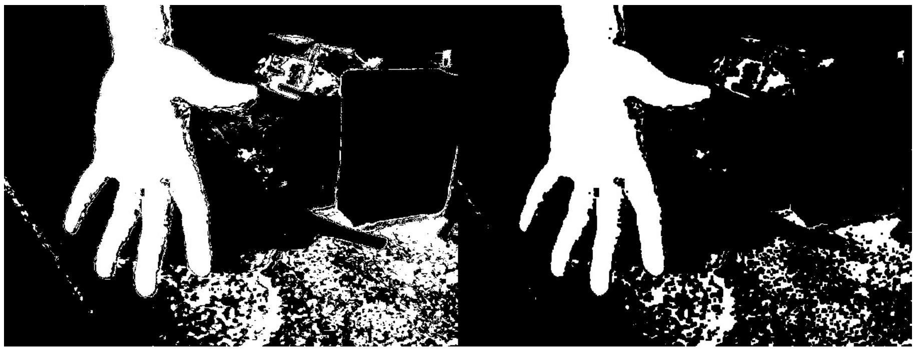


Figure 4.15: Erode filter.

Noisy isoleted areas can be removed by using an erode filter. The erode filter algorithm is the following:

- the maximum value is given only if the pixel already has the maximum value; also, the pixel needs to have at least a neighbor with the maximum value.

- zero value is given otherwise.

It means that isolated points or small noisy areas are deleted. Erode filter is applied because it reduces the calculation time of the contour retrieving algorithm described in the next subsection. In Figure 4.15 an example of erode filter is shown; you can see the original image on the left and the eroded image on the right. It is possible also to iterate the erode algorithm more than one time on the same binary image; this will remove the noise in a better way, but also it will affect the biggest area (tracked object area): it will become smaller. In order to bring back the original size to the tracked object area (the useful one), another algorithm is used: the dilate filter.

A dilate filter basically has the opposite function of an erode filter. In fact its purpose is to enlarge the white areas in the binary image. As already discussed, the erode filter is used to remove small areas noise, but also it affects the main object area, which is the important part of the image; the identified object area becomes smaller and so an other algorithm is necessary to bring it back to the original size. The dilate filter algorithm is the following:

- the maximum value is given to a pixel if it is already a maximum pixel, or if it has at least a neighbor with a maximum value. This is a dual condition to the erode filter; in fact, while in the erode filter there was an "and" condition, in the dilate filter there is an "or" condition, which is less strict;
- zero value is given otherwise.

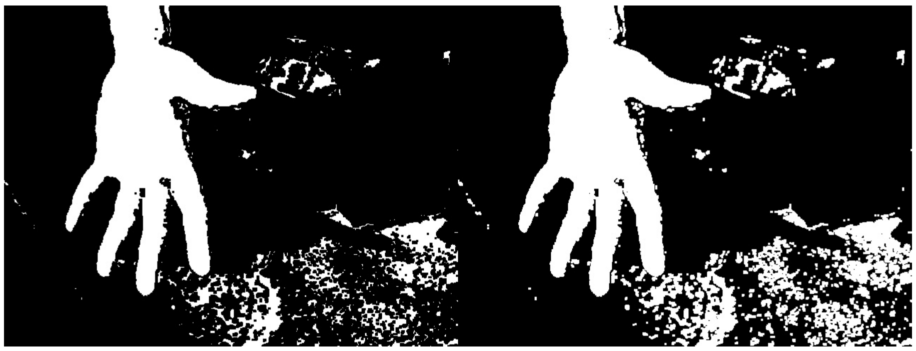


Figure 4.16: Dilate filter.

In Figure 4.16 an example of dilate filter is shown; from left to right: the original image and the dilated image. As you can see, white areas are bigger in the dilated image, because the contours are expanded. Comparing the initial image (on the left) in Figure 4.15 to the final image (on the right) in Figure 4.16 it is possible to appreciate the changing introduced by using both the erode and dilate filters: the hand size is the same, but some noise has been filtered (for example around the yellow box). However, some noise cannot be filtered and this is still a limit of algorithms based on color mapping. Even if the noise is not all filtered, following contour algorithms will do that; by using both erode and dilate filters, the number of contours in the binary image is reduced. This will affect positively the contours retrieving algorithms, so they will run faster and require less memory.

4.2.5 Contours retrieving

In the last subsection, erode and dilate algorithms have been discussed. As you can see in Figure 4.16 the result of both filtering is a binary image with more than one area. By the way, the only area needed is the hand area; only this area has to be taken into account. The OpenCV library already implements useful functions to retrieve the contour information from a binary image. Basically, the contour retrieving function implemented in OpenCV analyses the binary image and returns the contour datas in the image. As there might be many white areas in the binary image, the OpenCV function ("cvFindContours()") returns all the retrieved contours in the binary image; that's why it is necessary to interpose erode/dilate functions ("cvErode" and "cvDilate" OpenCV functions); since the number of contours is lower, the contours retrieving time gets smaller. Each contour information is given as an array of points, including all the points in a specified contour. These points are given according to the anti-clockwise orientation.

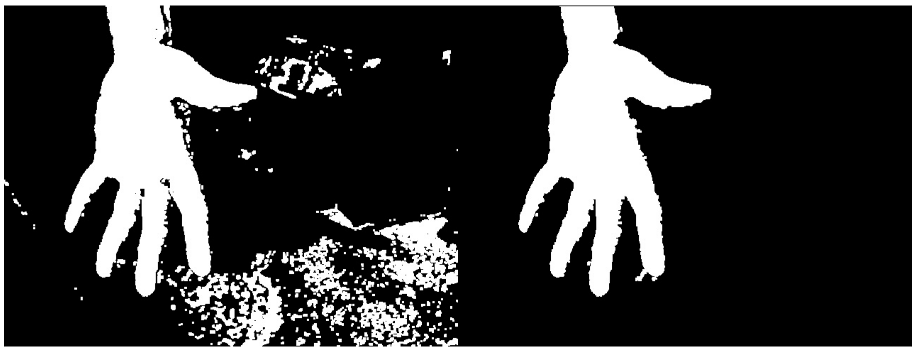


Figure 4.17: Contour retrieving.

After retrieving the contours informations, only the contour with the biggest area is taken into account. OpenCV already implements a usesul function to evaluate each contour area in terms of pixels ("cvContourArea"); the biggest area - the area with the maximum amount of pixels - is filtered. In Figure 4.17 only the biggest contour, i.e. the hand contour is considered. However, the hand contour is not so smooth, that's why an additional contour smoothing algorithm has been implemented.

4.2.6 Contours smoothing

As you can see in Figure 4.17, the hand contour looks noisy and needs to be smoothed by an additional smoothing function. The author implemented his own smoothing function, since it was not included in the OpenCV library. Basically, the algorithm is similar to a running average applied to the pixel coordinates of a contour. The used formulas are expressed below; the new x and y coordinates of a contour pixel are calculated as the mean value evaluating the points coordinates in a specified window; the new x(i) coordinate is calculated as the mean value of the x coordinate between (i-N) and (i+N); the same calculation method is applied to the y coordinate.

$$x'(i) = \frac{1}{2 \cdot N + 1} \sum_{k=i-N}^{i+N} x(k)$$

$$y'(i) = \frac{1}{2 \cdot N + 1} \sum_{k=i-N}^{i+N} y(k)$$

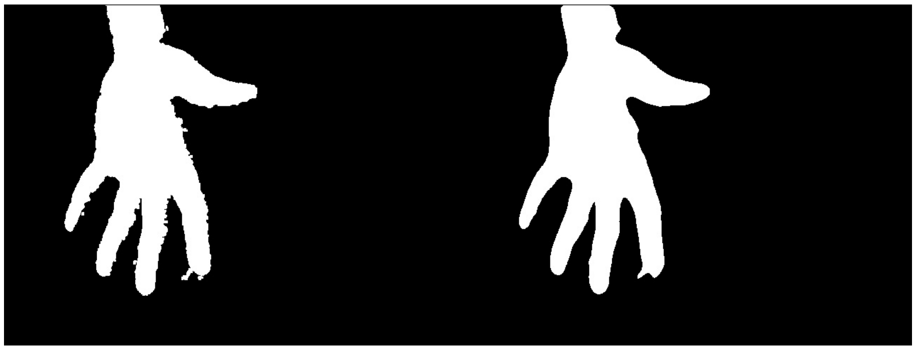


Figure 4.18: Contour smoothing.

For an image resolution of 640x480 pixels, a N parameter of 10 has been used. In Figure 4.18 the results of a hand smoothing has been reported. After obtaining a good smooth hand image, it is possible to search for the fingertips using the fingertip search algorithm described in the next subsection. This filter is easy, the computation cost is low and it has proven to be very useful.

4.2.7 Fingertips search algorithm

The fingertip search algorithm is applied to a contour and not to a binary image. Basically, it determines the maximum curvature points in a specified contour. As already explained, the maximum contour data are stored by OpenCV as an array of pixel coordinates in the image. These informations are filtered using the contour smoothing function, and then the fingertips search algorithm operates. In Figure 4.19 an example of hand image is shown; on the left side, the black shape represents a hand, while on the right side only the hand contour is shown.

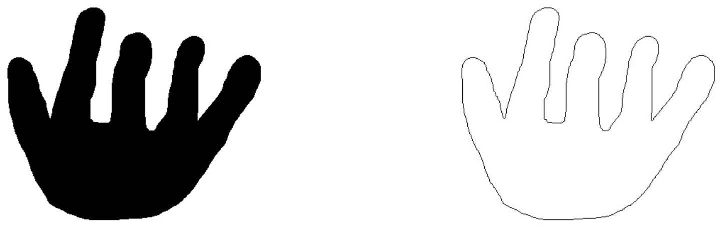


Figure 4.19: Fingertips search example.

For each point belonging to the contour, two vectors are calculated. In Figure 4.20 the calculated vectors are shown; there are two examples. Let us concentrate on the AB vector (called "u" from now on) and on the BC vector (called "v" from now on). These two vectors are standing from the middle point B (i-th point) to the point A, the (i+k)-th and C, the (i-k)-th. So, for each i-th point in the contour, two vectors are calculated: the next vector and previous vector. Then, the scalar product and the vector product (cross product) are calculated using the two vectors. Evaluating the angle (thresholding the angle) the algorithm finds the highest curvature points and evaluating the vector product, only the fingertips maximum curvature points are retrieved (positive curvature - hills, not valleys).

First of all, the angle between the vectors $\bar{AB}$ ($\vec{u}$) and $\bar{BC}$ ($\vec{v}$) is calculated. The angle is calculated using the following scalar product formulas. In this case, the cosinus of the angle between the two vectors is calculated. If the angle is higher than a minimum threshold - if the cosinus is lower than a threshold - the i-th point is considered a maximum curvature point candi-

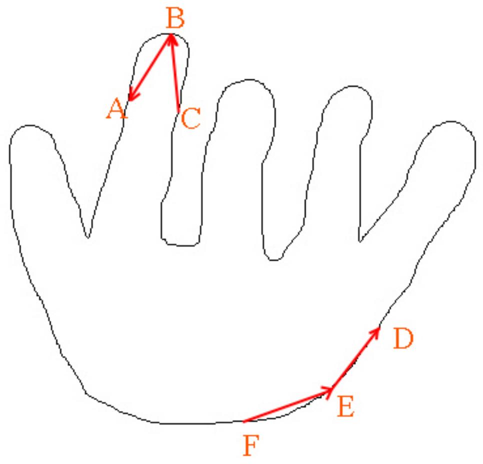


Figure 4.20: Vectors in fingertips search algorithm.

date. A good value for the angle is for example $\alpha = 100°$; this value has been obtained empirically.

$$\vec{u} = x_u\hat{i} + y_u\hat{j}$$

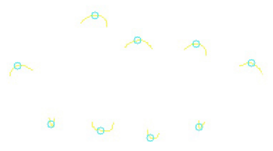
$$\vec{v} = x_v\hat{i} + y_v\hat{j}$$

$$\vec{u} \cdot \vec{v} = |\vec{u}| \cdot |\vec{v}| \cdot \cos\alpha$$

$$\vec{u} \cdot \vec{v} = x_u x_v + y_u y_v$$

$$cos\alpha = \frac{|\vec{u}| \cdot |\vec{v}|}{x_u x_v + y_u y_v}$$

In Figure 4.21, on the left side, all the maximum curvature candidates are shown in yellow color; for each group of pixels, the mean position pixel is taken as a fingertip point (azure color). However, an additional filter is required to filter only "hills" and not "valleys" fingertip candidates. This kind of filtering uses the vector product (cross product) between the two vectors. More specifically, only the orientation of the vector is evaluated, so the sign



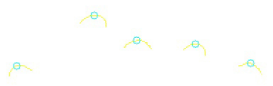


Figure 4.21: Found fingertips.

of the cross product. In fact:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_u & y_u & z_u \\ x_v & y_v & z_v \end{vmatrix} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_u & y_u & 0 \\ x_v & y_v & 0 \end{vmatrix} = \hat{k} \cdot (x_u y_v - y_u x_v)$$

By evaluating the sign of the cross product, it is possible to filter only the fingertip points. In Figure 4.21, on the right side, the result is shown. The fingertips search algorithm needs to be calibrated with two important parameters. The first one is the minimum angle α_{min} and the second one is the "k" parameter. While the minimum angle does not depend on the dimension (in pixels) of the tracked hand - good values are between 100° and 140° -, the k parameter depends on the dimensions of the hand. An empirical formula has been found by the author to determine the best k parameter depending on the hand area. In fact, if A is the hand total area (in pixels) and the hand is approximated as a square, the following formula applies $A = l^2$, where l is the side of the square. The value of the approximate side can be calculated as $l = \sqrt{A}$. An empiric percentage of l is used as the k parameter (from 20% to 50%). In fact, as the hand area becomes bigger, also the fingers become bigger and the k-parameter has to be changed. So, a good parameter for k can be the following.

$$k = c \cdot \sqrt{A}$$

with the following "c" constant value: $c = 0.2 \div 0.5$. In Figure 4.22 the found fingertips points are drawn with red circles.

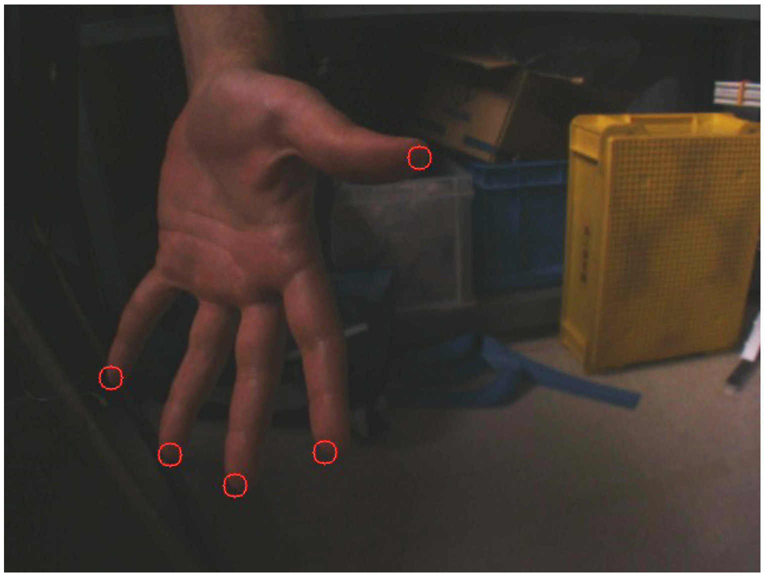


Figure 4.22: Hand with found fingertips.

4.2.8 Distance calculation using stereo vision

In order to calculate the distance of an object in a 3D space, it is necessary to obtain the position of the same object in both the left and right frames. That's why stereo vision requires two cameras to calculate the 3D position of an object. In this subsection, the method to calculate the distance of an object will be described. In Figure 4.23 and Figure 4.24 both the upper and lateral camera planes are shown; it is also possible to see the axis conventions used in the project. In particular, "y-axis" is the depth of the object, while x and z are respectively the width and height positions. The axis conventions can be seen also in Figure 3.1.

Basically, a function has been implemented by the author to calculate the distance of an object in a 3D space. The function has the following inputs and outputs:

- Inputs; the pixel position is given in both the left and right frames. So, the total number of inputs is four: x and y pixel positions in both left and right frames. Actually, the y position is redundant because due to a geometrical cause, the same object is seen at the same y position in both left and right cameras. If the two values of y differ in left and right camera frames, a mean value is calculated by the algorithm;

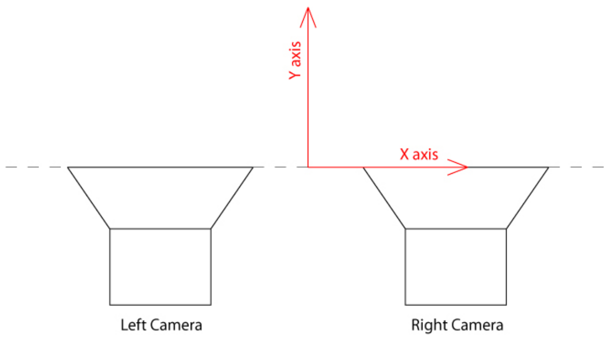


Figure 4.23: Camera axis, upper plane.

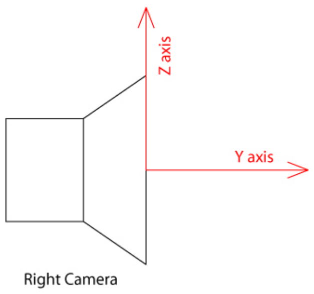


Figure 4.24: Camera axis, lateral plane.

- Outputs; evaluating the pixel positions in the left and right frames, the distance calculation algorithm gives the 3D position as an output (x, y and z axis positions).

So, there are 4 inputs and 3 outputs; the reason of that is because a y position is redundant; this means that the independant inputs are 3 instead of 4. From now on, the algorithm to calculate the 3D distance will be described. In Figure 4.25 an image showing the nomenclature conventions is shown. Each point has a 3D distance identified by the position in the three axis (x, y and z). First of all, it is necessary to calculate the distance in the y and x axis, and after that the calculation of z is very easy. The equations of the straight lines passing through the point P and the left and right lenses are the followings. The value of N_L and N_R are given in pixels, and they can vary in the range between $-R_x$ and $+R_x$.

$$y_L = \left(\frac{f}{s}\right)\Big|_x \left(\frac{R_x}{N_L}\right)\left(x_L + \frac{d}{2}\right)$$

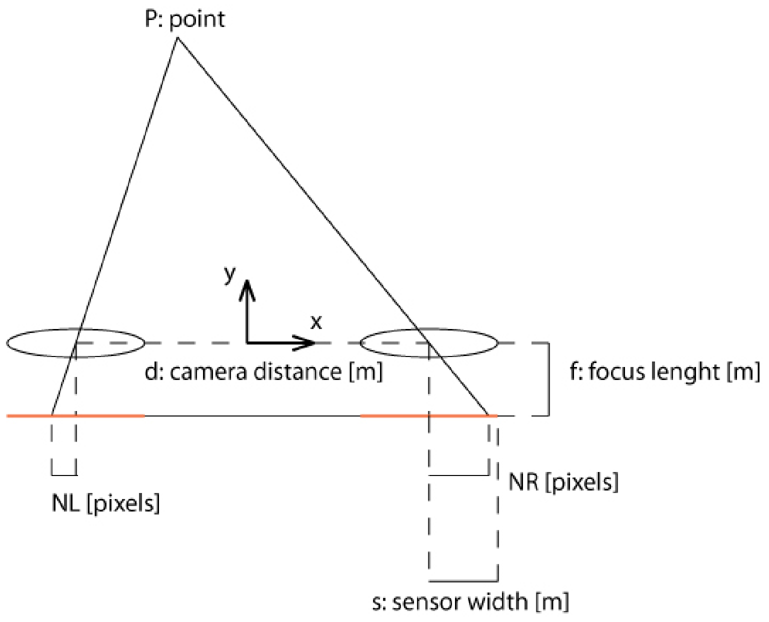


Figure 4.25: Stereo vision 3D calculation on x-axis and y-axis.

$$y_R = -\left(\frac{f}{s}\right)\bigg|_x \left(\frac{R_x}{N_R}\right)\left(x_R - \frac{d}{2}\right)$$

In the equations above, R_x is the semiresolution of the image on the x axis. For example, if the camera has a resolution of 640x480 pixels, the R_x parameter is equal to 320. Manipulating the equations, it is possible to calculate the x and y coordinates of the point, imposing the conditions $x_L = x_R = x_P$ and $y_L = y_R = y_P$. The (f/s) value on the x axis is the focal length on the x axis divided by the sensor semiwidth on the x axis.

$$x_P = \left(\frac{d}{2}\right)\left(\frac{N_L - N_R}{N_L + N_R}\right)$$

$$y_P = d \cdot \left(\frac{f}{s}\right)\bigg|_x \left(\frac{R_x}{N_L + N_R}\right)$$

After calculating the position in the x and y axis, the z position can be easily calculated with the following formula, according to the image in Figure 4.26.

$$z_P = -\left(\frac{f}{s}\right)\bigg|_z \left(\frac{H}{R_z}\right) \cdot y_P$$

In the formula above, the H value is given in pixels and it's related to the height position of the pixel in left and right images. In particular, H must be in the range between $-R_z$ and $+R_z$. In this case, R_z is the semiresolution in of the image on the z axis; for a 640x480 pixels image, $R_z = 240$.

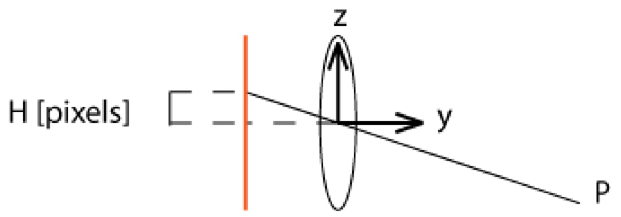


Figure 4.26: Stereo vision 3D calculation on z-axis.

4.2.9 Fingertips space matching and noise filtering

A method to find the fingertips in an image has already been explained in the last subsection. However, sometimes false detections can occur, as you can see in Figure 4.27; in the figure, on the left side, the fingertips retrieved by the algorithm are shown as red circles. In the left image, a false detect fingertip has been found; in fact, there's also a red circle on the forearm. An effective algorithm has been invented by the author of the thesis to filter many errors similar to this. The algorithm is based on the idea that each real fingertip has to be seen by both left and right cameras at the same pixel height plus or minus an accuracy amount of pixels. For example, if a fingertip is found in the left image at a pixel height of 190, the same fingertip has to be seen by the right camera, too, at a similar height (for example, from 185 to 195). If such fingertip is not detected in the right image, too, it means that there's an occlusion or maybe the fingertip detection is just caused by noise. So, a left fingertip is good only if there is at least one possible correspondance in the right frame.

More precisely, for each fingertip of the left frame, one or more correspondences are searched in the right frame. If no correspondence is found, the fingertip is not taken as a good fingertip and so it is deleted by the filter. If only one correspondence fingertip is found in the fingertip right image window, the 3D position can be calculated as already explained in the last subsection. But sometimes, each fingertip in the left frame has many candidates in the right frame, that's why a method to identify the best one is

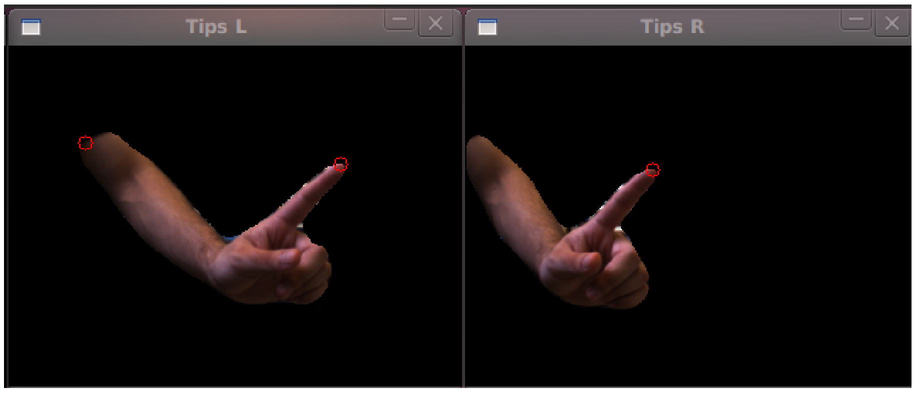


Figure 4.27: Fingertips false detects.

necessary. By consequence, the method proposed by the author searches the best correspondence fingertip in the right image, for each fingertip of the left image. The method is the following and it is applied for each fingertip on the left frame.

- First of all, the mean position of the hand area is calculated in both left and right frames. If the hands have similar area, it means that the areas are probably correct and they can be used. If the areas differ too much, it means that probably the tracked object is not a hand or that there might be some occlusion, and so the algorithm is not applied. Assuming that the areas are similar - i.e. the area difference is maximum 50%, for example - the algorithm starts. The mean translation of the hand between left and right frames is calculated. For example, if the hand center point is (200, 180) in the left image and (150, 185) in the right image, it means that the mean x-axis translation of the hand is 50 pixels. It must be noticed that the mean y position of the hand is similar due to geometrical cause (the two cameras are aligned).

- After the mean translation between the left and right frames has been calculated, the algorithm searches the best correspondence fingertip in the right image. However, the best correspondence is not searched in all the frame, but only in a small window centered in a specified point; the window central point has the same y coordinate as the right fingertip, and the x coordinate is the right fingertip x coordinate plus the mean hand translation. The window width and height can be changed by the user in realtime. In some cases, in the right search window, more than one fingertips are found. So, it is necessary to find a way to retrieve the "best" one. To do that, the OpenCV matching template function has been used: more specifically, the best right frame fingertip is the fingertip that has a histogram close to the left fingertip reference histogram. It means that if more than one fingertips are found in the search window in the right frame, their histograms are evaluated and only the most similar one is taken as the correct correspondence fingertip.

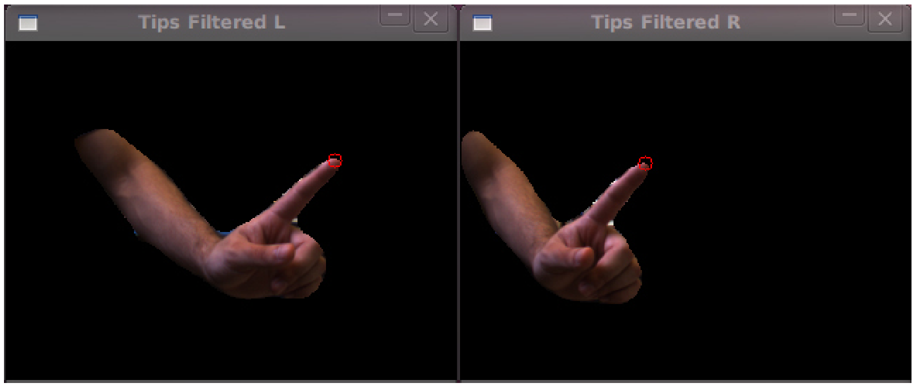


Figure 4.28: Fingertips matches after spacial noise filtering.

The algorithm described above is applied for each left fingertip. In Figure 4.28 the results of the spacial matching and filtering are reported; as you can see, the false detect of Figure 4.28 has been successfully removed. At the end of the filtering process, a new data structure is obtained: this new data structure contains all the fingertip correspondences found in left and right frames at a specified instant; in particular, for each correspondence, the coordinates of the fingertip in both left and right frames are stored. This structure is very useful because it easily permits to calculate the 3D distance using the algorithm introduced in the last subsection. However, not every false detects are deleted using this methods; in fact, it has been noticed that some false detects tend to appear and disappear, so also a time matching filter has been implemented by the author and it will be discussed in the next subsection.

4.2.10 Fingertips time matching and noise filtering

Sometimes, even the filter described in the last subsection is not filtering all the false detects. In facts, sometimes left noisy points have one or more matches in the right frame, and so the filtering described above is not functioning properly. To remove the residual noise, an additional time matching filter has been implemented by the author. Basically, the time matching filtering is similar to the space matching filter already described. The steps of the filter are the followings:

- At a specified instant, the hand mean position is calculated and the translation between the current and the previous position is computed; this means that the hand mean translation vector is calculated. As an approximation, we can say that the points in the previous frame has been translated by a vector that has a similar value to the hand mean translation vector.

- Based on the assumption that the hand pixels are translated by a specified vector that is similar to the hand mean translation vector (computed in the previous step), we can assume that each fingertip in the left image must have a correspondent previous fingertip in the frame before - i.e. in the frame previously captured. The time correspondent fingertip must be searched in a window centered in a specified (x, y) position; in particular, this window central position is the actual fingertip position minus the hand translation vector. So, for each left hand fingertip at the i-th instant, a fingertips correspondence is searched in the previous frame (i-1)-th in a specified window (the window has width and height specified by the user). If a matching fingertip is found in the previous frame, the fingertip "reliability coefficient" is increased.

- If more than one time correspondences are found in the search window, the fingertip which has the most similar histogram is taken as the best matching fingertip. The fingertip takes the same reliability of the previous and increses it by one. If no matches are found in the previous left frame, it means that a new fingertip has been found. This new fingertip is stored and a 1 reliability coefficient is assigned (it means that the fingertip has been found in only one consecutive frame).

The innovative approach of this time filtering is determined by the "reliability coefficient". This is a coefficient that stores the total number of times that a fingertip spacial correspondence has been found, so the total number of consecutive frames in which the same fingertip can be observed. If this number is high, it means that this tip is reliable and so it is not appearing and disappearing. But if the value is low (less than 5, for example), it means that the tip is still new and it could be both a good tip or a noisy tip.

The fingertip time matching feature can be used in this way: a minimum threshold reliability coefficient is set, for example 5. It means that if the same fingertip is seen for more than 5 consecutive frames, it is probably a good fingertip and so it is visualized. But if the value is below 5, the fingertip is filtered. This coefficient can also be called a "confidence coefficient" or "trust coefficient". If the coefficient is higher, the filtering is very effective. However, if the threshold is too high, when the fingertip is lost - due to occlusion, for example - it will take more time to recover the new position of the fingertip. During this downtime, the position of the fingertip has to be estimated with predictive methods or simply the last valid position can be taken constant.

Chapter 5

Project analysis and results

In this chapter, the written program will be analysed. After that, the results will be shown and then commented. In the first part of this chapter, the structure of the program is explained, and also the libraries used and the programming language. In the second part, the runtime results are reported: in particular, the logs of fingertips positions and calculation times.

5.1 Program analysis

In this subsection, the program structure is explained. The program has been divided into many parts which have specific functions. Each part of the program is analysed in detail: in the first part, general informations are given about the programming language and libraries used; then, the acquisition part of the program is explained: from the frame acquisition to the camera adjust and image filtering; the fingertip retrieving and matching are then analysed and, at the end, additional informations about the graphic user interface are given.

5.1.1 Program structure

The program has been written in C++ language using different editors, depending on the operative system: in Ubuntu Linux, Emacs and Anjuta have been used to write the code; in Mac OSX, X-Code has been used; Ubuntu Linux software does not require to pay for any licence, since it is a free software, while Mac OS-X operative system needs to be paid. In both operative systems (Ubuntu Linux and Mac OS-X) the program code has been compiled using the “g++” compiler including the used libraries. However, the finished program runs on a Ubuntu Linux system using open-source software, so no additional cost is needed for software.

The used language is C++, which is an object-oriented programming language. Unlike the C language, C++ permits the programmer to structure the software into different classes and divide the functions into many classes; this is useful because as the program becomes bigger, the functions of the program can be divided by areas and it is easier to the programmer to understrand the code and modify it. Additional libraries have been used: “libdc1394”, OpenCV and OpenGL.

- “libdc1394” is a library that is intended to provide a high level programming interface for application developers who wish to control IEEE 1394 based cameras that conform to the 1394-based Digital Camera Specification. The library development is an open-source project licensed under the GNU Lesser General Public License (LGPL). It was originally designed for the Linux platform but the new API version 2 is now also compatible with OSX.

- OpenCV is a library oriented to the computer vision. Originally it was developed by Intel; now, OpenCV is released under a BSD license, it is free for both academic and commercial use. It has C++, C, Python and soon Java interfaces running on Windows, Linux, Android and Mac. Many image processing functions are already implemented in OpenCV and can be easily used by other programmers. Also, OpenCV provides additional features such as graphic windows handling and also mouse and keyboard input handling.

- OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games. OpenGL is managed by the non-profit technology consortium Khronos Group.

Many classes have been declared, depending on the function. In particular, the declared classes are the followings. Each class contains attributes and methods, respectively variables and functions referring to the class.

- Class "SingleAcquisition". This class refers to all the attributes and methods regarding a single acquisition. As already explained, the physical system is composed by two IEEE 1394 cameras connected to a Firewire bus. Each "SingleAcquisiton" created object is related to a single camera, so in the program, two "SingleAcquisition" objects have been created: one refers to the left camera, and the second one refers to the right camera. Basically, this class contains the variables and functions needed for each single frame; more specifically, the improved functions are grabbing the frame, image filtering and the feature extraction; the feature extraction consists in using the color information to retrieve the hand and then filtering the contour and finding the "raw" fingertips in each hand and also other useful informations such as the hand mean point. So, this class contains all the attributes and methods that refer only to a single frame. Functions that are using both frames data are included in an upper class including two SingleAcquisition objects; this upper class is called "DoubleAcquisition". Below, the structure of the SingleAcquisition class is shown.

  ```
  class SingleAcquisition {
  public:
          SingleAcquisition(int camera);
          ~SingleAcquisition();
          // variables
          // functions
  };
  ```

- Class "DoubleAcquisition". This class refers to all the attributes and methods that are using both the left and right frames informations. A class "DoubleAcquisition" contains two object of class "SingleAcquisition", the left and right camera objects. The structure of the class is shown below and it will be deeply analysed in one of the following parts of this subsection. The functions in this class are related to the synchronized acquisition of both left and right frames and to the analysis of both left and right "raw" fingertips. In fact, the "DualAcquisition" class contains the functions that analyse both left and right "raw" fingertips and find the correspondences (fingertip matches), and provides both spacial and time filtering described in the last chapter. Moreover, this class contains the functions used to change both left and right cameras parameters at the same time.

```
class DualAcquisition {
public:
        DualAcquisition();
        ~DualAcquisition();
        SingleAcquisition* singleacquisitionL;
        SingleAcquisition* singleacquisitionR;
        // variables
        // functions
};
```

- Class “Graphics”. This class refers to the graphic interface. After the program useful algorithms have run, it is necessary to visualize the outputs in more windows. The class “Graphics” only contains the graphic elements related to the OpenCV windows. It means that this class deals with the OpenCV created windows, so it is necessary to plot the images with the identified fingertips and the filtered hand. But also it deals with the mouse input used to grab the reference hand. And also, it reads the keys pressed by the user while selecting the OpenCV windows. By the way, functions related to OpenGL are not declared in this class even if they’re related to graphics; in fact, they’re declared as global functions outside of the “main” function.

```
class Graphics {
public:
        Graphics();
        ~Graphics();
        // variables
        // functions
};
```

- Class “Program”. This class is the global class containing objects of all the other classes. The structure of the class is shown below. As you can see, the class contains variables related to the program operation. When the variable “start_track” is set to “true”, the tracking starts; this can be done only if a reference color histogram has been built. When the variable "exit_program" is set to "true", the program exits; this happens when the user presses the exit key. In the class "Program", two important objects are declared: the first one is a "DualAcquisition" object; basically, it manages both cameras, from the frame grabbing to the fingertips matching and filtering; the second one is the "Graphics" object that manages the OpenCV graphics. Also, some functions are included in the class: they're related to the search of reference files (used to build a histogram, when the program starts), to keys handling, to show help and so on. An important function is "evaluate_new_selection()": it uses a new selection made by the user to calculate a new reference histogram. The most important function of this class is "main_cycle()". When this function is called, the main cycle of the program is executed: the frames are grabbed, fingertips are extracted and matched, but also this function provides the handling for the user pressed keys. At last, there is a graphics linking function that is necessary to link the graphics images to the images in the "DualAcquisition" object; in other words, this function links the graphics part of the program with the image processing part of the program.

```
class Program {
public:
        Program();
        ~Program();
        bool start_track;
        bool exit_program;
        DualAcquisition* dualacquisition;
        Graphics* graphics;
        void search_files();
        void graphics_linking();
        void main_cycle();
        void evaluate_new_selection();
        void keys_handling();
        void key_help();
};
```

In Figure 5.1 the structure of the software is shown. Basically, the global sofware contains a "Program" object and also the OpenGL functions. When the program starts, an OpenGL function is called ("glutmainloop"); this function does the plot of the 3D window. The OpenGL main loop calls the "main_cycle()" function in the "Program" object, which does both the image processing part of the program but also it deals with the OpenCV windows. As you can see in the written code, the "main" function creates a

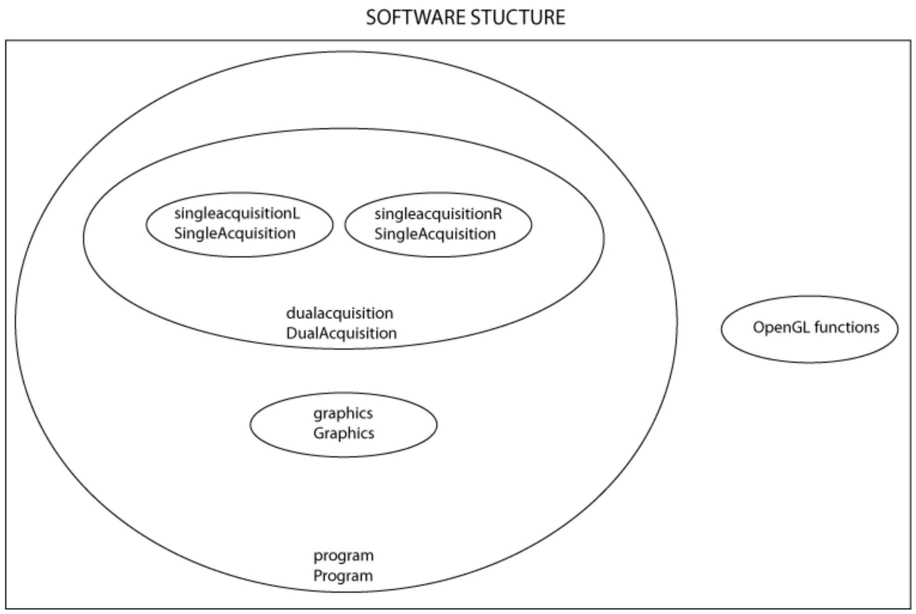


Figure 5.1: Software structure.

new "Program" object and initialises the OpenGL window. Then, after creating the OpenGL window, a display function is set; the "display" function plots the 3D axis, the cameras and the fingertips in a 3D window. Also, a keypressed function is set ("keyPressed") to manage the keys pressed by the user while using the OpenGL window. The idle function "idleFunc" basically call the "main_cycle()" function in the "Program" object, which does the image processing job. After the OpenGL setup, the "glutMainLoop()" function is called, which repeats the program cycle until the user wants to exit the program by pressing the exit program ("ESC" or "q").

```
int main(int argc, char** argv) {
        program = new Program;
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize (640, 480);
        glutCreateWindow("3D Axis Position");
        glutDisplayFunc(display);
        init();
        glutKeyboardFunc(keyPressed);
        glutIdleFunc(idleFunc);
        glutMainLoop();
        return 0;
}
```

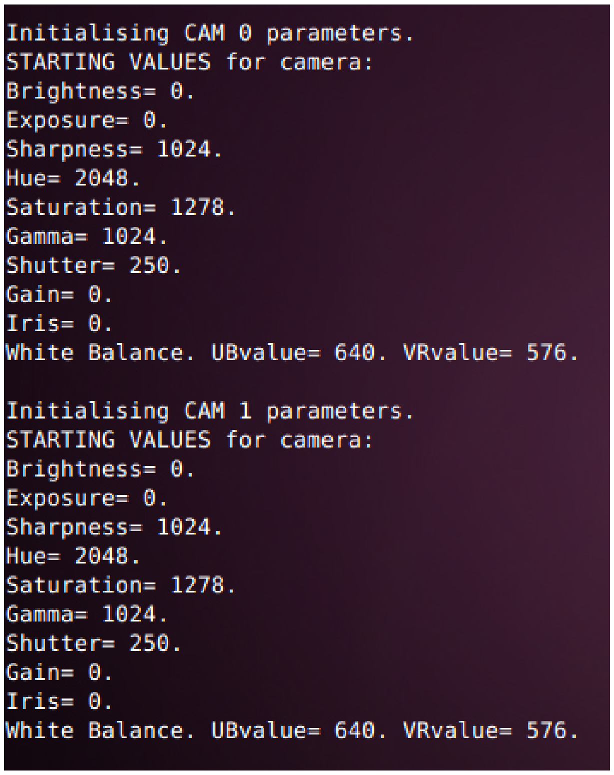
In the next part of this subsection, each part of the program will be deeply described. In particular, the frame acquisition and fingertip retrieving and matching processes will be explained, but also the opeation of the graphic user interface.

5.1.2 Frame acquisition and camera adjust

This part of the program is very important because the quality of the grabbed frames has a big impact on program operation. In fact, if the quality of the grabbed frame is not good, also the image processing implemented algorithms are not functioning well. This part of the program mainly does the following:

- camera adjust; the exposure parameters depend on the light condition of the environment. In particular, as the light becomes higher, the exposure time becomes smaller. It is very important to have a correct exposure because the image processing algorithms operate on the grabbed frame; if the image quality is good, the algorithms operate well on the images. So, it is necessary to have a good exposure and that can be obtained by setting the shutter time and signal gain of the camera. Normally, these parameters are set automatically by the camera if the autoexposure mode is on. However, the autoexposure function can cause problems to the program because the exposure parameters are not fixed but they are oscillating, and so they can affect the color filtering algorithm. To prevent this, the camera exposure parameters are fixed and they don't change automatically; by the way, the user can change them in realtime and this operation is recommended once before creating a new histogram. In fact, the camera exposure modify and the new histogram creation should be always performed together. If the exposure is not correct, for example if an overexposure is set, the camera sensor receives too much light and some photosites are saturated;

this means that their color becomes white and so the color convertion from RGB to HSV does not give correct informations about the color hue and saturation of a pixel. On the other hand, if the image is under-exposed, a similar problem occurs: the pixels grabbed by the camera are black and no color information can be retrieved about the hue and saturation. In the program, the user can increase or decrease the exposure by pressing the keys “a” and “z” respectively. It is also possible to change the saturation correction by using the “s” and “x” keys. The white balance parameters are mantained constant. The camera exposure parameters setting has been done using the “libdc1394” library. In Figure 5.2 the camera parameters setting messages are shown: this



```
Initialising CAM 0 parameters.
STARTING VALUES for camera:
Brightness= 0.
Exposure= 0.
Sharpness= 1024.
Hue= 2048.
Saturation= 1278.
Gamma= 1024.
Shutter= 250.
Gain= 0.
Iris= 0.
White Balance. UBvalue= 640. VRvalue= 576.

Initialising CAM 1 parameters.
STARTING VALUES for camera:
Brightness= 0.
Exposure= 0.
Sharpness= 1024.
Hue= 2048.
Saturation= 1278.
Gamma= 1024.
Shutter= 250.
Gain= 0.
Iris= 0.
White Balance. UBvalue= 640. VRvalue= 576.
```

Figure 5.2: Camera parameters setting.

message appears when the program starts, as the camera is initialised with standard values that can be modified by the user by pressing the beyboard keys.

- frame acquisition; in this project, the frame acquisition has a frequency of 15 fps (frames per second). This means that in a second, 15 frames are grabbed, so a new frame is grabbed every 66 ms. The number of grabbed frames can be increased to 30 fps or more, but there is a limit given by two factors: the first one is the exposure time, in fact each frame requires a minimum exposure time to be grabbed; the second one is the communication speed. Each frame requires a specified time to be transmitted from the camera to the personal computer, and the IEEE 1394b has a maximum bus speed (in this case, the masimum bus speed is 800 Mb/s). In this project, the frames have been grabbed by the camera using a RGB format (not compressed); each frame has a 640x480 pixel resolution and each pixel has 3 values (RGB), each one has a 8 bit resolution (1 byte). So, the number of bytes for each frame is the following.

  $$N_{bytes} = 3 \cdot 640 \cdot 480 = 921600\,{}^{\text{bytes}}/_{\text{frame}}$$

  Each frame requires approximately 1 MB to be transfered from the camera to the personal computer (8 Megabit). At each sample time, two frames have to be transfered from the cameras to the computer (left and right frames), so it means that the maximum sampling frequency that can be obtained with a 800 Mb/s bus is 50 fps (20 ms). To perform the grabbing of the images, the "libdc1394" library has been used. However, the image processing part of the program uses the OpenCV library, so the image data needs to be converted from the "libdc1394" format to the OpenCV format (IplImage format). After the image is grabbed using "libdc1394", it is converted to the OpenCV format and it can be used by the image processing algorithms.

After the image has been grabbed and converted to the OpenCV format, additional operations need to be performed on both left and right frames. It has been said that each frame is grabbed at a 640x480 pixels resolution. However, this resolution is too high to be used for image processing algorithms as it takes a lot of time to process the image at thit resolution. In order to reduce the calculation time, the image is scaled to half resolution: from 640x480, the image is scaled to 320x240 pixels; it means that the total amount of pixels in the scaled image is reduced by four, as both the height and width are reduced by two. After the image has been scaled to a new format, a mean filter is applied to reduce the image noise and apply some blur. The output image is then a filtered 320x240 image that is used for the following image processing algorithms. The frame acquisition is required to run the image processing algorithms, but also it is required to create the reference color histogram. This step will be described in the next part.

5.1.3 Histogram creation

A color histogram model is necessary since the image processing algorithms are based also on the color of the image. As already explained, the hand is filtered by using a color histogram model. Basically, a histogram is a probability distribution that contains the color data of a reference hand. The color histogram is built by evaluating the HSV color image; the HSV color image is obtained by converting the RGB color space information to the HSV color space. The histogram is a 2D histogram since only the Hue and Saturation planes have been evaluated (the Value plane has not been used).

To create a color histogram, it is necessary to get a high number of hand pixel and then use their color information to extract the color features and convert them to a probability histogram. The OpenCV library permits to create a histogram by using two images:

- one reference frame containing a hand; basically, it is a grabbed frame containing hand color information.
- a binary mask image; this image has the same resolution of the reference frames and each pixel has a zero value if it's not part of the hand and maximum value if it is part of the hand.

By giving both images as inputs to the histogram creation function, the function creates a 2D histogram using the Hue and Saturation planes. An example of histogram can be seen in Figure 4.11. In Figure 5.3 the left and right reference images are shown. In Figure 5.4 the left and right masks are shown. To create the histogram, both of them are used. In fact, the histogram creation function uses only the pixels belonging to the hand to create a color histogram. Two histograms are created: one for the left camera, and one for the right camera. In fact, even if the two cameras have the same exposure parameters, there might be some differences between the left and right camera sensors, and to prevent errors, two histograms have been built. As long as both left and right histograms have been created, the tracking

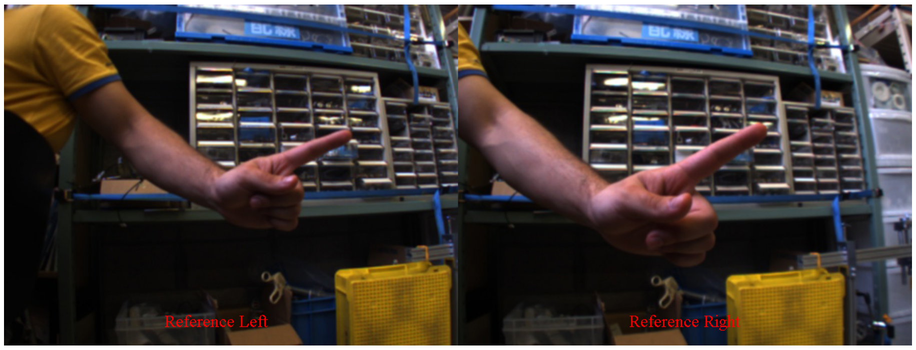


Figure 5.3: Histogram left and right reference images.

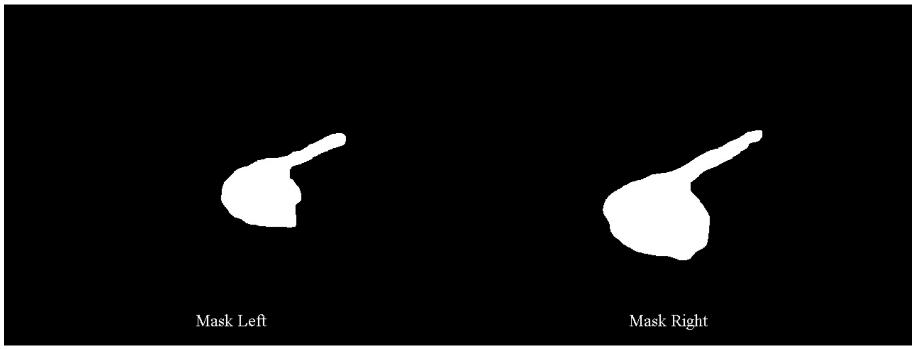


Figure 5.4: Histogram left and right mask images.

program can start. As you can see in Figure 5.5, the user selects the hand area in a frame by using the mouse. The user has to perform the selection two times: one for the left frame and one for the right frame. When the user is performing the selection, the last grabbed frame is stored and visualised until the selection is finished. In the figure, the green points indicate the points that has been passed by the mouse pointer; when the user releases the mouse button, all the points in the internal area (inside the green points) are taken as hand points. Then, both the reference frame and mask are passed to the histrogram creation function.

5.1.4 Fingertip retrieving

In the past subsections, the image acquisition and histogram creation have been explained. This subsection explains the algorithms used to extract the

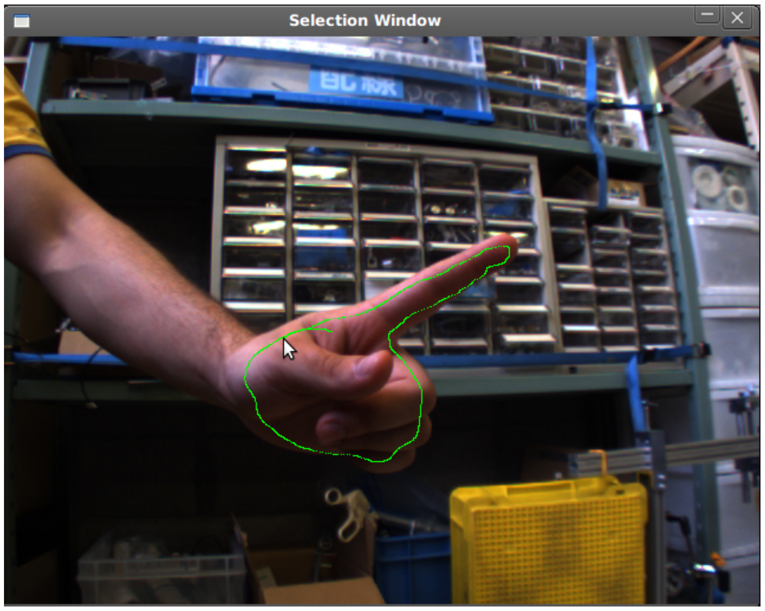


Figure 5.5: Histogram hand area selection.

fingertips informations from a single frame. Since only one frame is used to extract the tips positions, the fingertips found with method are called "raw" fingertips because some points can be noisy. In order to filter these noisy points, in the next subsection some implemented filters will be explained. So, the fingertip points calculated by using only one frame are just fingertip candidates that need to be filtered using both a spacial and time filtering.

As the single left and right frames have already been scaled and filtered, it is possible to use them for the following algorithms.

- conversion from RGB to HSV. The scaled and filtered frame is converted from the RGB color space to the HSV color space. This operation is performed because the HSV color space is more useful to identify the color of a pixel; in addition to that, the hue and saturation of a color are virtually independent of the strength of the environmental light. After the image has been converted to the HSV color space, only the Hue and Saturation planes are evaluated in the next step.

- back-projection image calculation. Evaluating the image Hue and Saturation planes and using the histogram informations, the back-projection image is calculated. As explained before, the back-projection is a probability image. It is a one layer image (greyscale image) in which every pixel has a value proportional to the probability to be a pixel belonging to the hand. You can see and example of back-projection in Figure 5.6; the left and right windows are respectively the left and right frames back-projection images.

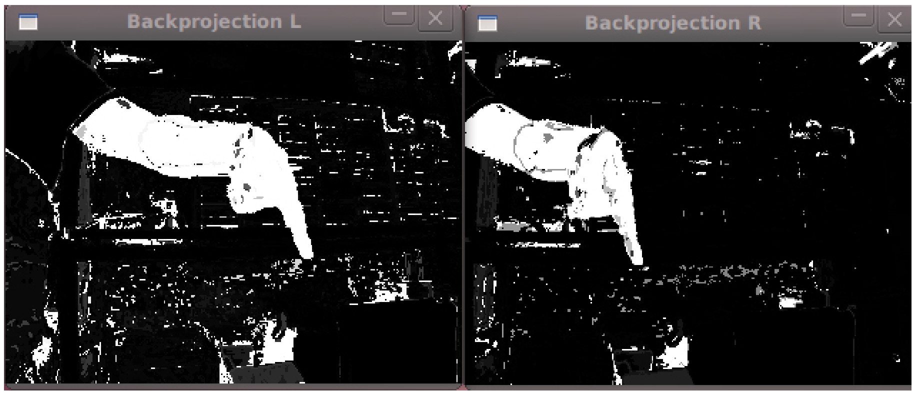


Figure 5.6: Back-projection calculation.

- back-projection thresholding. The back-projection image is thresholded with a specified constant parameter. The threshold level has been put to 128. In fact, each pixel has a 8 bit color resolution (values from 0 to 255), so 128 is a half scale value. After the thresholding operation, a binary image is obtained. In the obtained binary image, each pixel with the positive value is a pixel belonging to the hand, while pixel with a 0 value are pixels not beloning to the hand. The threshold parameter can be adjusted by the user in realtime.

- erode and dilate filters. On the hand binary image, consecutive erode and dilate filters are applied to remove small noisy areas. The total number of erodes and dilates can be changed in realtime by the user. By increasing the erode value, the noise is removed more effectively, but the details of the hand contour and also an other problem can occur: sometimes it happens that the hand area splits into two more areas, and this affects the following algorithm that is used to take only the biggest area. So, if the erode value is too high, part of the hand could bel lost.

- maximum contour retrieving. In the binary image, all the contours are identified and after that only the biggest one is taken - i.e. the contour with the biggest internal area, in term of pixels amount. The biggest area is considered the hand area. However, the hand contour is not smooth and so a smoothing filter is required.

- contour filtering. The smoothing filter is basically a running average filter and it has been explained in the previous chapter. By changing the number of pixels used in the filtering window, the strength of the filter becomes higher but also the contour details are lost. This number can be changed in realtime by the user. At the end of this step, the hand area is obtained. From now on, the fingertip retrieving algorithm operates. In Figure 5.7 you can see the filtered hand area obtained; the filter effectiveness is evident as the hand is very smooth.

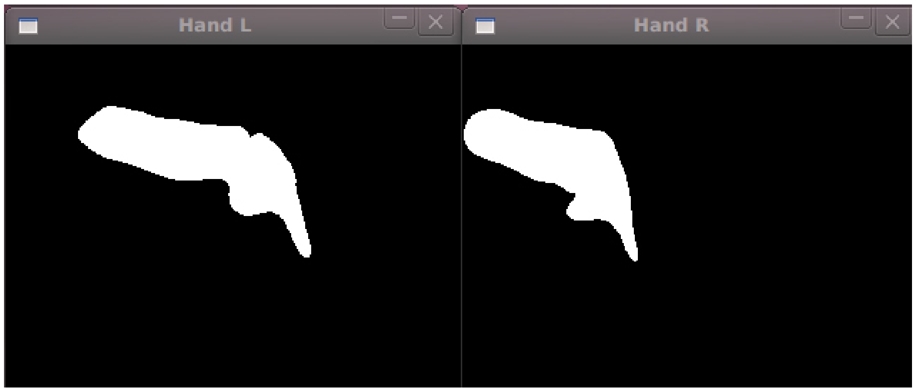


Figure 5.7: Hand area.

- fingertip search. As the contour has been smoothed by a smoothing filter, it is possible to calculate the fingertip candidates in the contour. This method has already been deeply explained in the last chapter. Basically, the fingertip retrieving method is based on finding the highest curvature points in the hand contour. However, maximum curvature points can be both “hills” or “valleys”; the only interesting points are hills, as they correspond to fingertips, while valleys correspond to the space between two fingers. So, the fingertips are obtained by using both the scalar product and cross product. In Figure 5.8 you can see the tips found by the fingertip search algorithm. As you can see, the total number of fingertips found in the left and right frames is four; two fingertips are correct, while the other two fingertips are noisy points. In fact, if you see Figure 5.7, in the hand area there are some high curvature areas that are taken as fingertips even if they are not. Because of the presence of noisy points, it was necessary to find out a way to remove this noisy points. In the next subsection, the method used to filter the noisy points is explained.

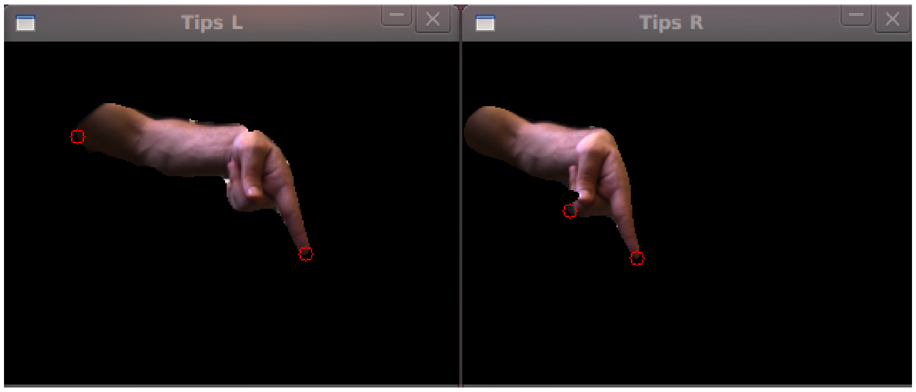


Figure 5.8: Fingertips not filtered.

5.1.5 Fingertip matching

A method to find the fingertips candidates has been explained in the last subsection. However, as you can see in Figure 5.8, this algorithm is not perfect and sometimes also noisy points are taken as fingertips even if they are not. The presence of noisy points is caused by the presence of high curvature areas that are not fingertips, but they are still detected by the fingertip search algorithm, because the algorithm is based only on the curvature of the contour and not also on other informations. In fact, as you can see in the figure, both on the left and right frames, additional points are found because in the contour their curvature is high. In this subsection, a method to match the left and right fingertips will be explained. Also, this method has the purpose to remove some noisy false detected fingertips.

The method used to find the correspondences in the left and right images has already been explained in the last chapter. It is based on the fact that each correct fingertip candidate has to be seen at the same height by both left and right cameras. Since the cameras are aligned, each point in the 3D space is seen at the same height in both left and right frames. This information is very useful, because when finding fingertips, we don't need to analyse all the combinations between left and right fingertip candidates. In fact, for each fingertip in the left image, the fingertip matching algorithm searches for a possible correspondence fingertip in the right image. But the correspondence is not searched in all the frame, but only in a restricted area. This search area has a rectangular shape, because the correpondence finger can be only in specified height and width ranges.

More specifically, let us call $\begin{bmatrix} x_L & y_L \end{bmatrix}$ the position of a fingertip candidate in the left frame. Before searching for possible matches, the algorithm calculates the x-axis translation vector of the hand area between the left and right frames. In fact, due to geometrical causes, the same hand is seen into different places in the left and right frames; evaluating the left and right frames, after extracting the hand area, the hand mean position is calculated in both left and right frames; the hand mean position has two coordinates (x-axis and y-axis in the image - width and height pixels). After calculating the hand mean points in the left and right frames, a translation vector is calculated; by the way, the only information needed is the x-axis translation of the hand mean point; let us call this value x_T, as it is a translation on the x-axis given in number of pixels. The mean translation value is a very important number, because we can do the following approximation: each point of the hand in the left image has to be seen in the right image in a position related to the left position; in particular, the height position (image y-axis) needs to be the same - empirically, it has been noticed that there is always some pixels uncertainty; the x-axis position of the fingertip in the right image is the left position plus the translation value. So, the right probable position is $\begin{bmatrix} x_R & y_R \end{bmatrix} = \begin{bmatrix} x_L + x_T & y_L \end{bmatrix}$. Of course, the fingertip cannot be searched only in one point but it must be searched in a specified window centered in the position written above. However, the width and height are not specified and they have been obtained empirically. In the program, the user can change the height and width search parameters in realtime. As these parameters become smaller, the effectiveness of the filter becomes higher, but also some fingertip matches can be lost; by the way, if these parameters are too high, the effectiveness of the filter is low.

When searching for fingertip matches, in the right window there might be more than one matching candidates. For example, in the search window, there could be both the correct fingertip and also one or more noisy points. In order to discriminate the best one from the others, a coefficient must be used. In this particular case, the best match is found by evaluating the histograms of each possible match. More specifically, for a specified fingertip candidate in the left frame, the histogram is created (using a small window, for example a 10x10 pixels window); the left tip histogram is then compared with the histograms of the possible fingertip matching points in the right frame. The correct fingertip match is the match that ensures the highest coefficient of similarity between the two histograms (left and right tips histograms).

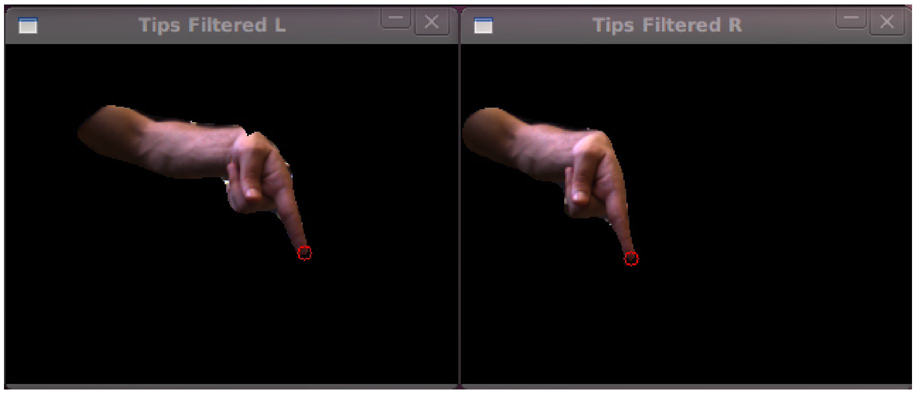


Figure 5.9: Fingertips filtered.

This fingertip matching method if not only a matching method but also a filtering method, because if a fingertip candidate in the left image does not have at least one possible correspondent fingertip in the right image, it is not taken. This method has proved to be effective, as you can see by comparing Figure 5.8 with Figure 5.9. In fact, the two noisy fingertips in the first figure don't have a correspondence fingertip in the other frame; for example, the noisy fingertip candidate in the left frame in the forearm position does not have a correspondent matching point in the right image. And also the noisy point in the right image is not corropondent to a point in the left image, so it is not taken. This filtering method is very useful and it has been called "spacial filtering" as it uses both the left and right frames grabbed at the same time. However, sometimes a noisy point in the left image has a matching point in the right image; this happens because hand contours have high curvature areas that are seen by both left and right cameras. By consequence, sometimes also noisy fingertips cannot be filtered by using this "spacial filtering" method.

To filter the residual noise, an additional filter has been implemented by the user and it has been called “time filtering”. In fact, the “spacial filtering” evaluates the data of two frames grabbed at the same time but in different positions (left and right frames). On the other hand, the “time filtering” evaluates two frames grabbed by the same camera but at different sampling times. More specifically, the time filtering implemented in this project evaluates the left frames grabbed at the i-th and (i-1)-th sampling times. The “time filtering” algorithm operation is similar to the “spacial filtering” algorithm. In fact, let us define the position $\begin{bmatrix} x_i & y_i \end{bmatrix}$ of a fingertip belonging to a fingertip correspondence at time i-th in the left frame. The mean position of the left hand in frame i-th and (i-1)-th is calculated, and the translation vector $\begin{bmatrix} x_t & y_t \end{bmatrix}$ between the two positions is then calculated. For each fingertip spacial correspondence, the left fingertip is taken at the i-th frame. Then, a window in the (i-1)-th frame is considered; this window is centered in the position $\begin{bmatrix} x_{i-1} & y_{i-1} \end{bmatrix} = \begin{bmatrix} x_i - x_t & y_i - y_t \end{bmatrix}$ and it has height and width specified by the user and empirically determined. The algorithm searches for possible matches in the (i-1)-th window; the method is similar to the method already explained for the “spacial filtering”. In fact, it is necessary to identify a correspondence between a fingertip in the i-th frame and (i-1)-th frame to understand the evolution of a fingertip position in the time. If more candidates are found in the search window, only the candidate with the most similar histogram is taken.

The method described above is very useful to filter the noisy tips that appear and disappear in just some frames. The basic idea is to give a “reliability” value to each fingertip spacial correspondence. This value becomes higher if the same tip is seen in many consecutive frames. More specifically, if a left fingertip at the i-th frame has a time correpondence with a fingertip at the (i-1)-th frame, it takes his reliability value and increments it by one. If a fingertip does not have a time correspondence, his reliabitity is set to one, as the fingertip is seen for the first time. Only the fingertips spacial correspondances which have a reliability value over a specified threshold are taken as valid. For example, if the threshold is set to 5, a fingertip needs to be seen for more than 5 consecutive frames to be considered a reliable fingertip. At each sampling time, all the fingertips over the threshold are plotted in the 3D window, but only the fingertip with the highest reliability coefficient is considered the tracked fingertip.

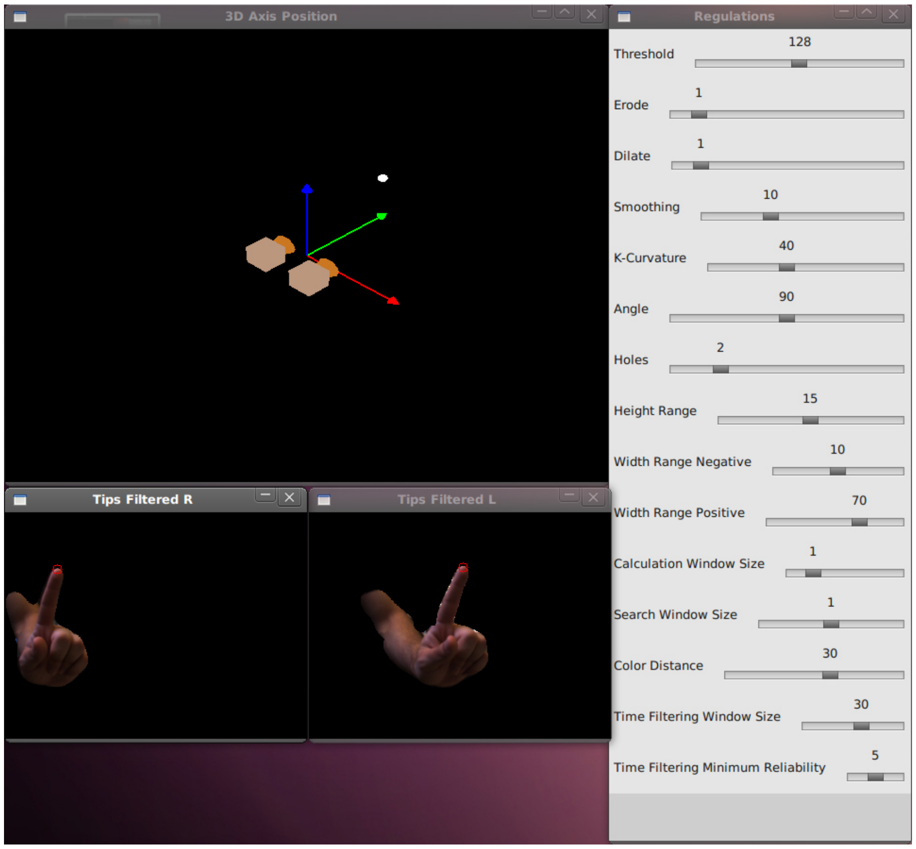
5.1.6 Graphical user interface

One of the most important parts of the program is the GUI (Graphical User Interface). The GUI is useful because it permits the user to interact with the program to read the results and to change the program parameters in realtime. The Graphical User Interface has been divided in two parts, because their purpose is different and also different libraries have been used to create the GUI:

- OpenCV related GUI. This part of the Graphic User Interface is using the OpenCV library and is related to the visualisation of the grabbed frames and also to processed images. OpenCV easily permits the programmer to create windows and to use them to visualise the images stored in the memory. In this library, the images are stored with an "IplImage" format, which is a structure containing data and information on the related image. The OpenCV library is also used to handle the keyboard inputs by the user, for example to increase or decrease the camera exposure time and saturation correction and to quit the program. The third purpose of the library is to handle the mouse inputs on the "selection window". In fact, the user needs to select the hand area used to calculate the color histograms for both left and right cameras. A useful function has been added to the program to improve the calculation speed: by pressing a key, it is possible to hide the less useful windows, so that the calculation speed is increased. The last purpose of the OpenCV library is to handle the user parameters regulation. In fact, a window provides many sliders that can be modified by the user to change the program parameters in realtime.

- OpenGL related GUI. The OpenGL library has been used to create a 3D window used to plot the filtered fingertips. The 3D window contains also a simple representation of the environment (the two cameras are plotted) and also the 3D axis are plotted, so it's easy to understand the real position of the fingertips in the real environment. In the 3D window, OpenGL plots the fingertips as spheres with different colors: the white and biggest sphere is the fingertip with the maximum reliability; the smaller and coloured speres represent the fingertips that have a reliability above the threshold but are not maximum reliability fingertips.

Also, additional informations are reported in the terminal. For example, when the user changes the camera parameters by pressing a key, the new parameters are reported in the terminal; and when selecting a new histogram, the correctness of the selection is reported and written in the terminal. By pressing a key, it is also possible to show additional program messages given in real time, such as the 3D position of each fingertip and the number of tip correspondences found at each sampling time.



Threshold 128
Erode 1
Dilate 1
Smoothing 10
K-Curvature 40
Angle 90
Holes 2
Height Range 15
Width Range Negative 10
Width Range Positive 70
Calculation Window Size 1
Search Window Size 1
Color Distance 30
Time Filtering Window Size 30
Time Filtering Minimum Reliability 5

Figure 5.10: Program with regulations.

In Figure 5.10 you can see the program main windows - i.e. the most important windows in the program. The shown windows are the following:

- 3D window. The biggest window shows the position of the fingertips in a three dimensional space. The axis xyz have been plotted and there is also a simple representation of the two cameras. Fingertips are represented with spheres: the white and biggest one represents the fingertip with the maximum reliability, while the smaller and colored spheres represent the fingertips which have a reliability over the threshold but that are not maximum reliability tips. However, in the screenshot, only the white sphere is present.

- Filtered tips in left and right images. These are the fingertips obtained by the fingertip search algorithm after being filtered by the spacial filtering. In the graphical interface, both left and right hands fingertips are plotted. In these two images, only the hand recognised area is shown and red circles represent the fingertips after being filtered.

- Regulation window. An other window lets the user change the program parameters at runtime. There window contains sliders which can be changed, and so it is possible to modify a program parameter while the program is running. These parameters are related to the program algorithms. With a runtime parameter regulation, the user can easily modify the parameters and also understand the influence of each parameter in the program operation.

The windows that can be seen in the figure are only the most important windows in the program. In fact, these windows show only the final results to the user. Visualising few windows can be useful as it increases the software calculation speed. Without many windows to be plotted, the cycle time is reduced. By the way, if necessary, it is also possible to plot more windows. By plotting all the windows, the calculation time becomes higher but it's possible to see the program operating steps one by one. For example, we can check the correct operation of the backprojection by checking the backprojection window. In Figure 5.11 you can see the program windows: all program steps are visualised.

- 3D window. It's the same as in Figure 5.10. In this window the three dimensional position of the fingertips is shown.

- Backprojection windows. Two windows are showing the backprojection image. As already explained, the backprojection image is a grey scale image in which every pixel has a specified value proportional to the probability to be a hand pixel.

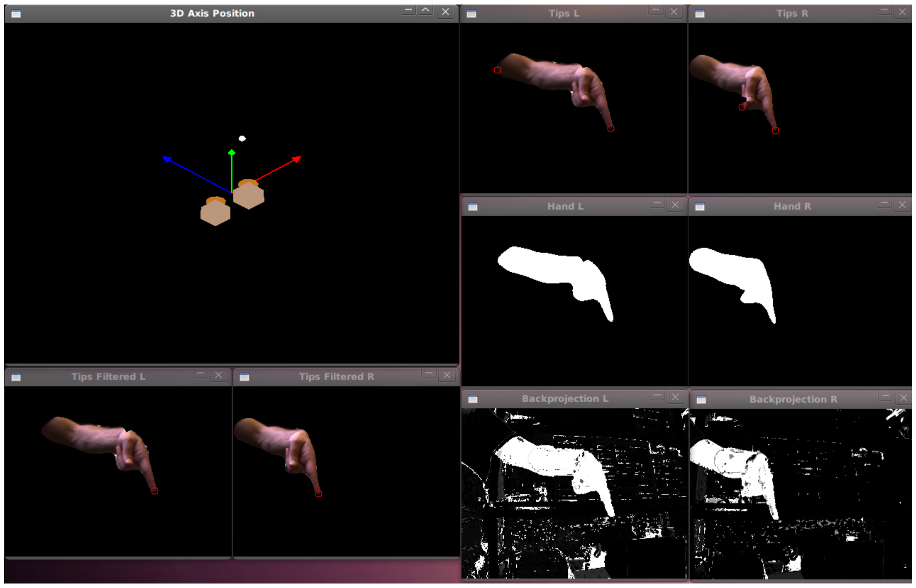


Figure 5.11: Program with all windows.

- Hand contour windows. Two windows are showing the hand areas in both left and right frames. The hand plotted in each image is obtained by thresholding the backprojection image and then applying the additional filters that has been explained in the past sections: erode and dilate filters, maximum area filtering and contour smoothing.
- Tips without filtering. These images are showing the fingertip candidates found by using the fingertip search algorithm. The fingertip search algorithm uses the maximum contour information to find the maximum curvature points and then plots these points in the image. These tips are obtained just by evaluating the contour curvature, so some points are noisy, as you can see in the figure.
- Tips after spacial filtering. These two images show the fingertips found after a spacial filtering. As you can see, some noise is removed.

Two different modes are provided by the program: the first one shows only the windows in Figure 5.10; this interface is light because there are less windows to be shown and so the calculation time is less. The interface in Figure 5.11 shows all the windows and it's useful if it is needed to visualise the program operation step by step. For example, if the user wants only to track the fingertip, the light interface is better; on the other hand, if the user wants to test the influence of a parameter to the program operation and see what happens step by step, the second interface is better because it shows all the windows. Changing the interface mode is easy and the user just needs to press the "w" key to switch from the light interface to the other and viceversa.

One useful part of the program is the regulations window which can be seen in Figure 5.10. The program parameters that can modified by the user are the followings:

- Threshold. This parameter is the threshold value used for thresholding the backprojection image. By using the backprojection image which is a greyscale image, a binary image is obtained. The default value in the program is 128, as it is a half range value between 0 and 255.

- Erode. The total number of erode filters applied to the thresholded image. The default value is 1. If this number is high, the small noisy areas are removed with more effectiveness, but also the main hand area can be splitted into two or more areas if the value is too high; if the hand area is splitted into more areas, a problem occurs, since only the biggest one is taken. So, a good value for this parameter is 1. If the parameter is set to zero, no filter is improved.

- Dilate. The total number of dilate filters applied to the eroded image. The default value is 1. Normally, this number should be the same as the "erode" number. In fact, if the total number of erodes and dilates are the same, the hand area is filtered without modifying its size; if the dilate parameter is higher than the erode parameter, the hand area is enlarged more than what it's eroded, so it becomes bigger; if the dilate parameter is less than the erode parameter, we obtain the opposite effect of reducing the hand size. That's why the "erode" and "dilate" values should be mantained equal.

- Smoothing. This is the smoothing factor. As already explained in the last chapter, when filtering the contours, each component in the contour array has a x-axis and a y-axis coordinate in the image. This filter simply calculates the x and y mean values in a specified window which has the dimension specified by the smoothing value. So, if the smoothing value is increased, the effectiveness of the filter increases, but also the details of the hand are lost. If the filter is too strong, the fingertips are smoothed too much and they're not retrieved. The default value is 10.

- K-curvature. This parameter is used by the fingertip search algorithm, and basically it's the distance used to calculate the two vectors in the algorithm. If the k-curvature parameter is higher, the starting and ending point of each vector are far. However, as already explained in the last chapter, this value is a percentage and the real value depends on the hand area dimensions. In fact, given a hand area, the square root is calculated - i.e. an approximated lenght dimension in pixel; by multiplying the k-curvature percentage, we obtain the k-curvature in pixels. After some empirical experiments, a value of 40 (40%) has proved to be a good value.

- Angle. This parameter is used by the fingertip search algorithm and corresponds to the minimum angle between the two vectors. If the angle is over this thresold value, the pixel is considered part of a fingertip. The default value is 90°, but also values between 80 and 130 degrees are good. If this number is too small, also noisy points are taken as fingertips. If this value is high, some fingertips may not be found by the algorithm.

- Holes. This parameter is used in the fingertip search algorithm. Basically, sometimes it happens that in the same fingertip not all the points are connected, because some points don't have an angle over the threshold; by consequence, the same hand fingertip can be splitted in two fingertip candidates. To prevent this, a "hole" value is used. More specifically, if there are some pixels above the threshold, they're still taken as belonging to the fingertip if they have fingertip points neighbors. For example, let us suppose that in the contour we have 10 pixels above the threshold, 2 pixels under and other 13 pixels above the angle threshold. If the "hole" value is more than 2, all the points are considered belonging to the same fingertip. However if the "hole" value is below 2, for example one or zero, the same fingertip is splitted in two fingertips and the same phisical fingertip is identified by two fingertip candidates instead of one. The default value is 2.

- Height range. This is the height (in pixels) used to implement the spacial filtering. More specifically, it is half height of the search window in the right frame. The default value is 15 pixels and it means that the search window has a total height of 31 pixels (15x2+1 pixels).

- Width range negative. This parameter is related to the height of the search window. The default value is 10 pixels.
- Width range positive. This parameter is related to the height of the search window. The default value is 70 pixels. More specifically, the sum of the "width range negative" and "width range positive" plus one is the height of the search window.
- Calculation window size. When finding the fingertip correspondences between the left and right frames, a useful additional algorithm has been implemented. Basically, the used fingertip position is recalculated evaluating the near points. The points with the most similar color histograms are taken as correspondent points. So, this calculation window size correponds to the size of the window used to calculate the histogram. The default value is 1, so the calculation window has a 3x3 pixel size, as both the height and width are 1x2+1 pixels.
- Search window size. This parameter is the size of the search window used to modify the fingertip correspondence coordinates. The default value is 1, so the search window size is 3x3 pixels.
- Color distance. This parameter is used when evaluating the histograms. If the histograms are too different, the point correspondence is not taken as valid. The default value for this parameter is 30, and it's proportional to the difference between the two histograms in left and right frames.
- Time filtering window size. This is the size (used both for height and width) of the search window, in the time filtering algorithm. The default value is 30.
- Time filtering minimum reliability. This parameter is very important because it is the reliability thresold used to decide if a fingertip can be considered reliable or not. At each time, only the fingertip correspondences that have a reliability value over the threshold are considered correct. The default value is 5. If this value is high, the effectiveness of

the filter is higher, but also many problems occur. For example, if the number is too high and the fingertip is lost, the recovery time is very high.

The graphics interface is used also to select the hand area needed to calculate the reference color histogram. As you can see in Figure 5.12, the user performs a selection using the mouse. The points passed by the mouse pointer are shown in green color. When the user releases the left mouse button, the area inside the points passed by the mouse is considered the hand area. Evaluating the color of the points in the area, the color reference histogram is then calculated.

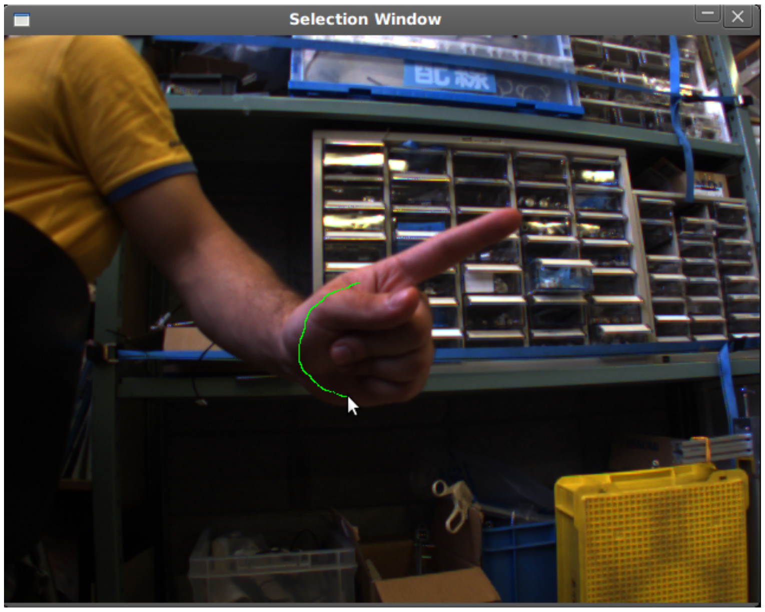


Figure 5.12: Manual selection of the hand area.

5.2 Results

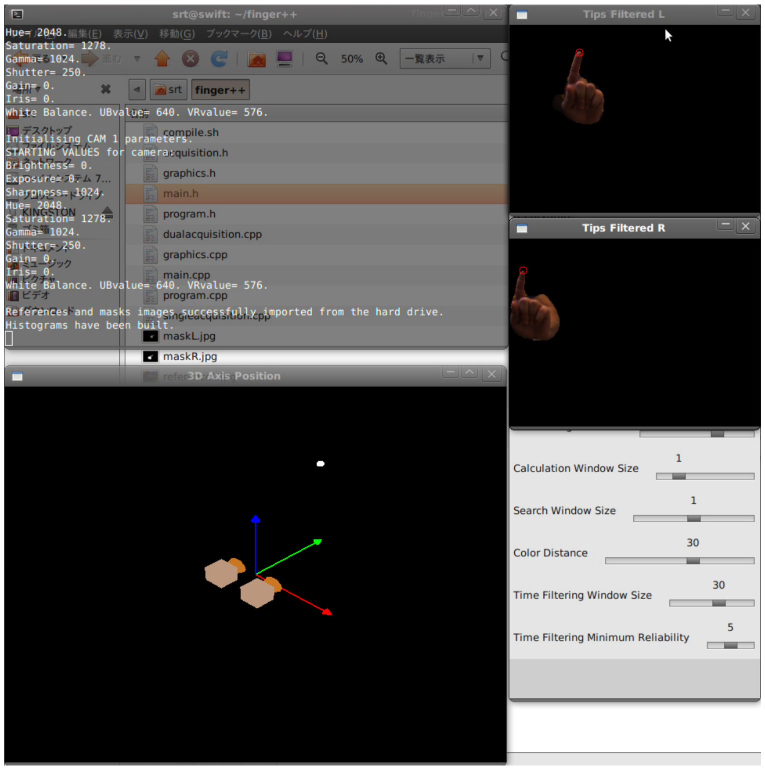
In this section, some results will be reported and then commented. In particular, two experiments have been done to measure and evaluate the performance of the program: in the first experiment, the user has a standing finger in front of the camera; in this case, the position of the fingertip is retrieved and the noise is analysed. In the second experiment, the finger is moving in the air making some rotations; in this second case, the interesting part is about the loss of the fingertip position: the fingertip position is lost and it takes some time for the program to get the new position. Both cases will be analysed in the following subsections.

5.2.1 Fingertip standing still

The first experiment that has been done is about a fingertip standing still in front of the two cameras. The author has put his finger in front of the cameras and the finger is not moving in the space. This experiment has been done to measure the static performance of the program, i.e. the noise created by the fingertip tracking algorithm. The position at every moment has been tracked by the software and then exported to a “txt” file. After that, the data file has been imported in Matlab to plot the position in a 3D space and to measure the noise and calculate the statistics values. In Figure 5.13 the experimental setup is shown: as you can see, the user has put the fingertip right in front of the camera and the fingertip position is found in each frame. In Figure 5.14 the position log has been reported. Each row of the “txt” file logs the position of the maximum realiability fingertip at a specified instant. In each row, the logged values are the following:

- Number of the fingertip. Each fingertip time correspondence has a number.
- Realibility value. This is the reliability value for each fingertip. The minum has been set to 5 (program standard value).
- x-axis position. This is the axis position given in meters.
- y-axis position. This is the axis position given in meters.
- z-axis position. This is the axis position given in meters.
- Time stamp. The time stamp is given in seconds: the count starts when the program starts.

The camera sampling frequency is 15 fps (frames per second), so a new frame is grabbed every 66 ms. To filter the 3D position, for each axis, a linear discrete time filter has been used. The filter is basically a moving



```
Hue= 2048.
Saturation= 1278.
Gamma= 1024.
Shutter= 250.
Gain= 0.
Iris= 0.
White Balance. UBvalue= 640. VRvalue= 576.

Initialising CAM 1 parameters.
STARTING VALUES for camera:
Brightness= 0.
Exposure= 0.
Sharpness= 1024.
Hue= 2048.
Saturation= 1278.
Gamma= 1024.
Shutter= 250.
Gain= 0.
Iris= 0.
White Balance. UBvalue= 640. VRvalue= 576.

References and masks images successfully imported from the hard drive.
Histograms have been built.
```

Calculation Window Size 1
Search Window Size 1
Color Distance 30
Time Filtering Window Size 30
Time Filtering Minimum Reliability 5

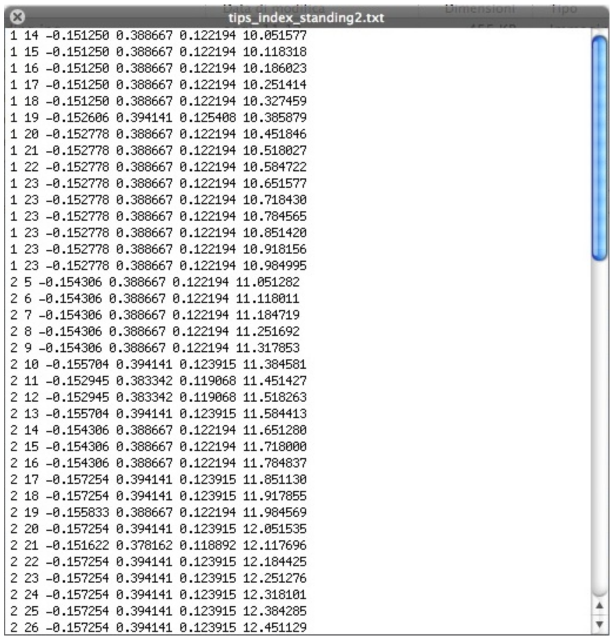
Figure 5.13: Fingertip standing still.

average filter with a specified window. In this thesis, two moving average filters have been used and they will be compared: the first one is a 2 values running average, the second one is a 10 values running average. The filter effectiveness of the second filter is higher but also the time delay is higher. The filters operate in the following way. Depending on the value of N, the effectiveness of the filter changes.

$$y_N(n) = \frac{1}{N}\sum_{k=0}^{N-1} u(n-k)$$

The following formulas are the equations of the two used filters. The first one uses a 2 values window, the second one uses a 10 values window.

$$y_2(n) = \frac{1}{2}\left(u(n) - u(n-1)\right)$$



```
1 14 -0.151250 0.388667 0.122194 10.051577
1 15 -0.151250 0.388667 0.122194 10.118318
1 16 -0.151250 0.388667 0.122194 10.186023
1 17 -0.151250 0.388667 0.122194 10.251414
1 18 -0.151250 0.388667 0.122194 10.327459
1 19 -0.152606 0.394141 0.125408 10.385879
1 20 -0.152778 0.388667 0.122194 10.451846
1 21 -0.152778 0.388667 0.122194 10.518027
1 22 -0.152778 0.388667 0.122194 10.584722
1 23 -0.152778 0.388667 0.122194 10.651577
1 23 -0.152778 0.388667 0.122194 10.718430
1 23 -0.152778 0.388667 0.122194 10.784565
1 23 -0.152778 0.388667 0.122194 10.851420
1 23 -0.152778 0.388667 0.122194 10.918156
1 23 -0.152778 0.388667 0.122194 10.984995
2 5 -0.154306 0.388667 0.122194 11.051282
2 6 -0.154306 0.388667 0.122194 11.118011
2 7 -0.154306 0.388667 0.122194 11.184719
2 8 -0.154306 0.388667 0.122194 11.251692
2 9 -0.154306 0.388667 0.122194 11.317853
2 10 -0.155704 0.394141 0.123915 11.384581
2 11 -0.152945 0.383342 0.119068 11.451427
2 12 -0.152945 0.383342 0.119068 11.518263
2 13 -0.155704 0.394141 0.123915 11.584413
2 14 -0.154306 0.388667 0.122194 11.651280
2 15 -0.154306 0.388667 0.122194 11.718000
2 16 -0.154306 0.388667 0.122194 11.784837
2 17 -0.157254 0.394141 0.123915 11.851130
2 18 -0.157254 0.394141 0.123915 11.917855
2 19 -0.155833 0.388667 0.122194 11.984569
2 20 -0.157254 0.394141 0.123915 12.051535
2 21 -0.151622 0.378162 0.118892 12.117696
2 22 -0.157254 0.394141 0.123915 12.184425
2 23 -0.157254 0.394141 0.123915 12.251276
2 24 -0.157254 0.394141 0.123915 12.318101
2 25 -0.157254 0.394141 0.123915 12.384285
2 26 -0.157254 0.394141 0.123915 12.451129
```

Figure 5.14: Standing finger position log.

$$y_{10}(n) = \frac{1}{10}\sum_{k=0}^{9} u(n-k)$$

In Figure 5.15 the absolute value and the angle of the filters transfer functions have been plotted; the two transfer functions have different colors and you can see that the "10 filter" has a higher effectiveness in removing the high frequency noise but also the phase delay is higher. In Figure 5.16 the position in the three axis has been reported: for each axis, the blue points represent the points calculated by the software without using a discrete time filter; the continuous red line represents the position in the three axis after the filter has been used; in this case, the "2 filter" has been used. As you can see, adding the "2 filter" does not remove the noise. These are the statistics calculated using the program time series (not filtered series). The followings are the mean values.

$$x_{mean} = -0.1552\,m$$

$$y_{mean} = 0.3871\,m$$

$$z_{mean} = 0.1211\,m$$

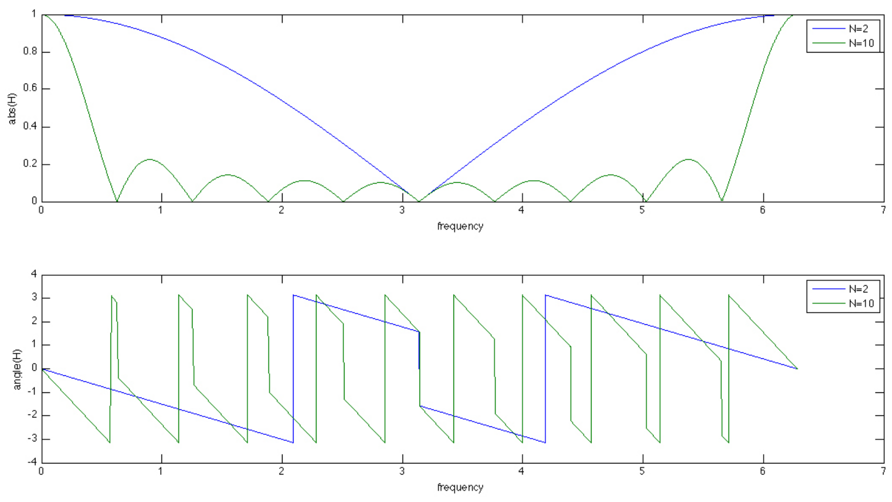


Figure 5.15: Filters transfer functions.

The following values are the standard deviations of the time series. As you can see, the standard deviations are about 2 mm on the x-axis, 5 mm on the y-axis and 2 mm on the z-axis. The highest standard deviation is on the distance value, in fact y-axis represents the depth; so, the noise filtering is necessary expecially for evaluating the distance of the fingertip.

$$x_{stddev} = 0.0023\,m$$

$$y_{stddev} = 0.0046\,m$$

$$z_{stddev} = 0.0020\,m$$

In Figure 5.17 a filter has been applied using the “10 filter”. As you can notice, the effectiveness of the filter is high because the output signal is quite stable. It must be noticed that some noise can be caused by the user hand movements; in fact, even if the user hand is standing still, there could be some noisy movements or some drifts in the position. By using the “10 filter”, much noise is removed: so, this kind of filter is good to filter the noise when a fingertip is standing still; however, as the filter order becomes higher, also the time delay becomes higher and so this kind of filter can introduce delays when the position has to be actuated by an actuator. In Figure 5.18 two 3D graphs are shown: on the left side, the 3D position without filtering has been shown, while on the right side the 3D position with the “2 filter”

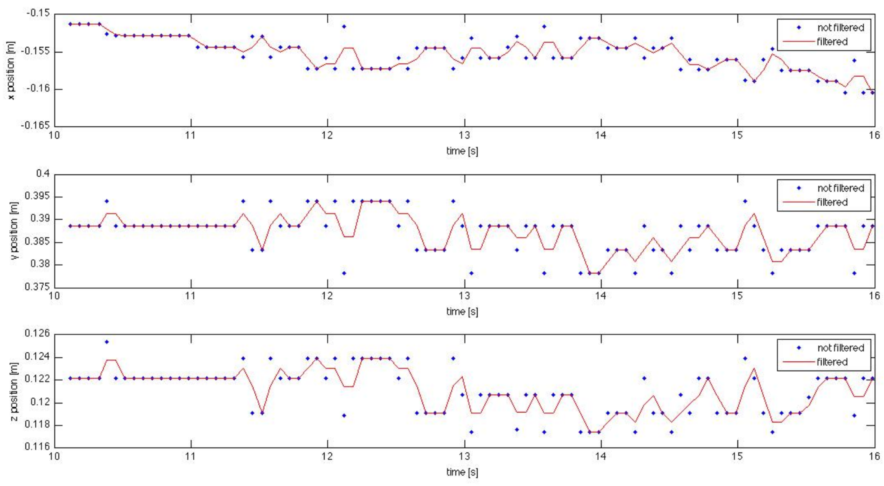


Figure 5.16: Position in the three axis using the "2 filter" (standing).

has been shown. As you can notice, the "2 filter" does not introduce many changes to the signal: the signal still remains noisy. In Figure 5.19, a similar graph is plotted: on the left side, the 3D graph plots the position of the fingertip in the three axis without filtering; on the right side, the fingertip position has been filtered using the "10 filter". As you can see, the noise is removed in a better way.

As already explained, many filters can be used to remove the noise from the position calculated by the software. In this subsection, only simple filters have been used, based on simple moving averages; also, many other filters can be implemented. Some considerations can be done about the use of filters:

- filters are useful since they can remove the noise from the software calculated position. This target is necessary because the positions must be sent to a mechanical actuator; if the signal is too noisy, the actuator may have problems with saturations.

- by increasing the order of the filter, the effectiveness becomes higher; it means that the noise is strongly removed because high frequencies in the signals are cut.

- increasing the order of the filter also has bad consequences, as it introduces a high delay that can affect the execution of the movement. This

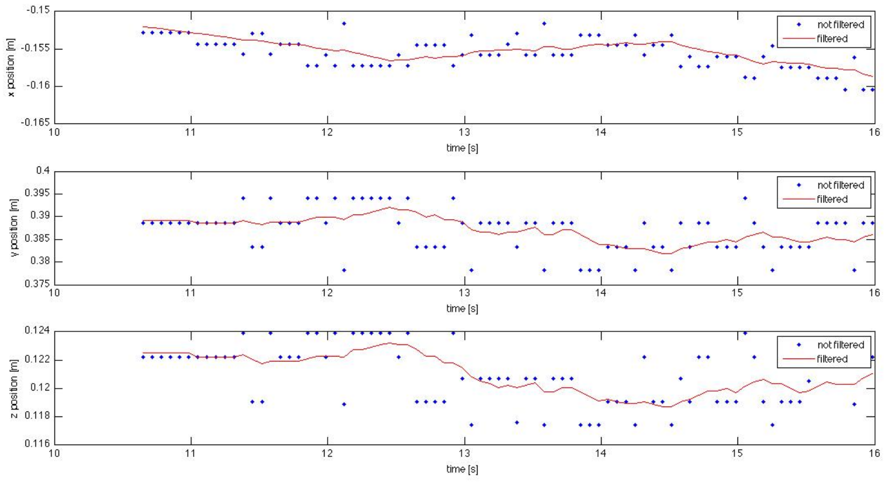


Figure 5.17: Position in the three axis using the "10 filter" (standing).

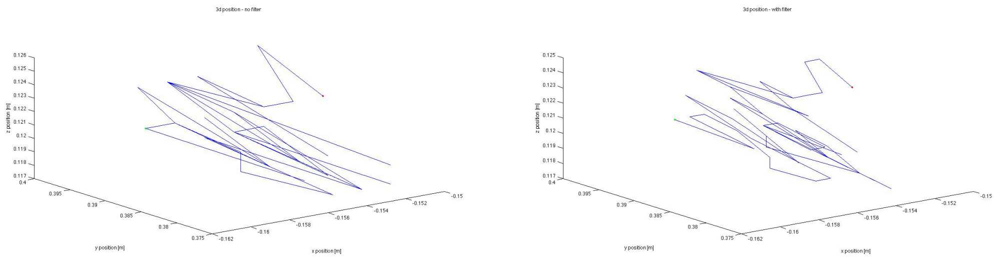


Figure 5.18: Fingertip standing 3D position with "2 filter".

aspect will be analysed in the next subsection.

As you can see in the axis position of Figure 5.16, the noise is higher in the depth (y-axis) and it's lower in the x-axis and z-axis. Because of this, it could be possible to apply different filters depending on the axis. For example, we can apply a "10 filter" to the y-axis and a "2 filter" to the x-axis and z-axis. This method can be good since it does not introduce delays to signals that doesn't have much noise: some time delay is introduced to the y-axis (depth) but not to the other two axis.

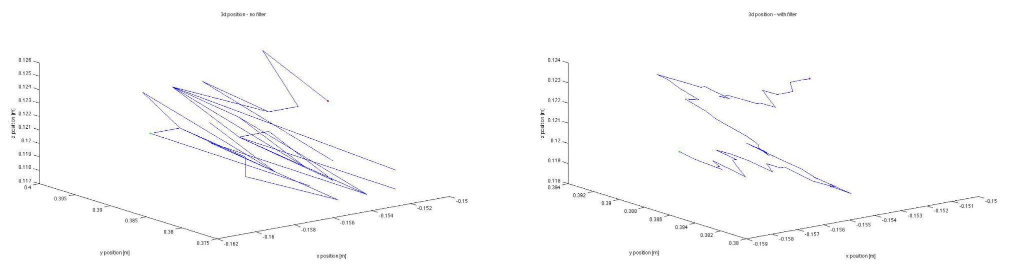


Figure 5.19: Fingertip standing 3D position with “10 filter”.

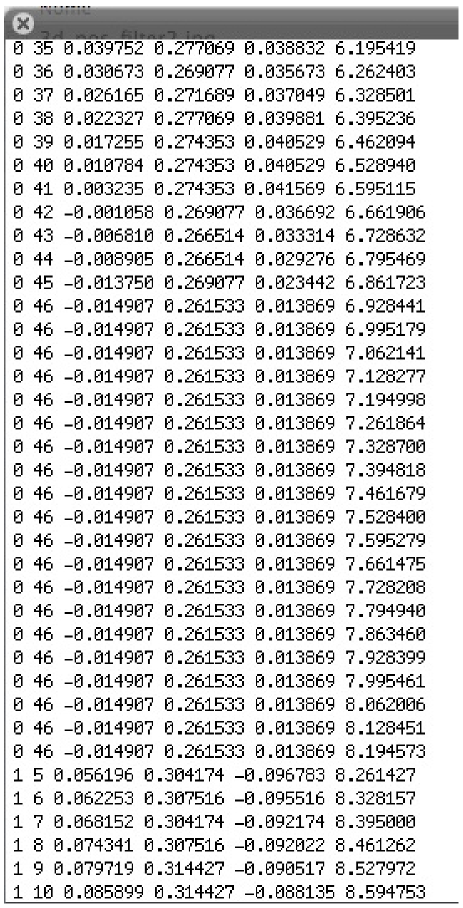
5.2.2 Fingertip rotating

The second experiment refers to a fingertip rotating in the three dimensional space. In this experiment, the user has put his fingertip in front of the two cameras and the movements are rotation movements in the space. In the last subsection, the purpose of the experiment was to measure the noise introduced by the program. In this experiment, the purposes are the followings:

- study the case of fingertip position loss. Sometimes, due to objects occlusion or because of the noise, the fingertip position is lost. It has been explained that when the position is lost, it takes a specified number of frames to get the new position. This number of frames is equal to the minimum reliability - i.e. the reliability threshold. For example, if a fingertip position is lost for one frame and the reliability threshold is equal to 5, it takes 5 frames to obtain the new fingertip position. In fact, the program considers a fingertip “reliable” only if the same fingertip is seen for more than 5 frames consecutively. This is to remove noisy points that appear for just some frames and then disappear; this algorithm has been called “time filtering”. In this experiment, when the position is lost, the last reliable value is taken constant. In fact, no predictive algorithms have been implemented.

- study the noise suppression using filters. Also in this case, the same two filters have been used. The first one is the “2 filter” and the second is the “10 filter”. The transfer functions of these filters have already been shown in Figure 5.15. As you can see in this figure, the effectiveness of the “10 filter” is higher when removing the noise, but also the time delay introduced could be too high, expecially if the movement must be actuated very quickly. The filters are useful also in the case of position loss, because they can smooth the axis position graphs.

From now on, the results of the experiment will be discussed. In Figure



```
0 35 0.039752 0.277069 0.038832 6.195419
0 36 0.030673 0.269077 0.035673 6.262403
0 37 0.026165 0.271689 0.037049 6.328501
0 38 0.022327 0.277069 0.039881 6.395236
0 39 0.017255 0.274353 0.040529 6.462094
0 40 0.010784 0.274353 0.040529 6.528940
0 41 0.003235 0.274353 0.041569 6.595115
0 42 -0.001058 0.269077 0.036692 6.661906
0 43 -0.006810 0.266514 0.033314 6.728632
0 44 -0.008905 0.266514 0.029276 6.795469
0 45 -0.013750 0.269077 0.023442 6.861723
0 46 -0.014907 0.261533 0.013869 6.928441
0 46 -0.014907 0.261533 0.013869 6.995179
0 46 -0.014907 0.261533 0.013869 7.062141
0 46 -0.014907 0.261533 0.013869 7.128277
0 46 -0.014907 0.261533 0.013869 7.194998
0 46 -0.014907 0.261533 0.013869 7.261864
0 46 -0.014907 0.261533 0.013869 7.328700
0 46 -0.014907 0.261533 0.013869 7.394818
0 46 -0.014907 0.261533 0.013869 7.461679
0 46 -0.014907 0.261533 0.013869 7.528400
0 46 -0.014907 0.261533 0.013869 7.595279
0 46 -0.014907 0.261533 0.013869 7.661475
0 46 -0.014907 0.261533 0.013869 7.728208
0 46 -0.014907 0.261533 0.013869 7.794940
0 46 -0.014907 0.261533 0.013869 7.863460
0 46 -0.014907 0.261533 0.013869 7.928399
0 46 -0.014907 0.261533 0.013869 7.995461
0 46 -0.014907 0.261533 0.013869 8.062006
0 46 -0.014907 0.261533 0.013869 8.128451
0 46 -0.014907 0.261533 0.013869 8.194573
1 5 0.056196 0.304174 -0.096783 8.261427
1 6 0.062253 0.307516 -0.095516 8.328157
1 7 0.068152 0.304174 -0.092174 8.395000
1 8 0.074341 0.307516 -0.092022 8.461262
1 9 0.079719 0.314427 -0.090517 8.527972
1 10 0.085899 0.314427 -0.088135 8.594753
```

Figure 5.20: Rotating finger position log.

5.20 you can see the position of the maximum reliability fingertip at each grabbed frame, i.e. at each sampled time. These are just some values of the time serie, but it is possible to evaluate these values and make some considerations. The number conventions are the same that have been used in the last subsection; in order, the following values are shown: fingertip number, fingertip reliability, x-axis position, y-axis position, z-axis position, time stamp.

- Two fingertip numbers are logged: the first one is 0, the second one is 1. This happens because at a specified time, the position of fingertip 0 is lost. When the program start, the first tracked fingertip is the fingertip 0. Its reliability coefficient increases at each frame, starting from 5; by the way, in the log, the starting value is 35.
- Starting from 5, the realiability of the fingertip 0 increases at each grabbed frame, because in each frame a time correspondence is found.
- When the realiability reaches the value 46, the fingertip is lost. In this case, the position loss is not caused by occlusion but it's caused by other factors. As you can see, at time 6.928 s, the last reliable position is found. In the next frame, at time 6.995 s, the fingertip position is lost.
- Since the fingertip position has been lost at time 6.995 s, the last reliable position is considered valid and it's mantained constant until time 8.194 s. So, the position is mantained constant for more than one second.
- At time 8.261 s, the fingertip position is tracked again; in fact, the fingertip realiability is 5 (the minimum value, the reliability threshold). Starting from this time, the reliability coefficient increases by one at each frame.
- When the fingertip is lost, the number of the next fingertip is different since the program, when evaluating the fingertip 1, does not find the fingertip 0 in the previous frame, because the fingertip 0 is lost for one or more frames.

In Figure 5.21 the position of the fingertip in the three axis has been reported. The fingertip is rotating in the space. As you can see in the graphs, the points calculated by the program are plotted with blue points, while the red line corresponds to the filtered signal using the "2 filter". At time 6.995 s the position of the fingertip is lost and so the position is mantained constant in the three axis. When the position is found again at time 8.194 s, the signals have a big discontinuity. The discontinuities are the followings: x-axis position varies from -0.014 m to 0.056 m, y-axis position varies from 0.261 m to 0.304 m, z-axis position varies from 0.013 m to -0.096 m. So, the instant variation of the position is about 60 mm on the x-axis, 40 mm on the y-axis and 110 mm on the z-axis. Probably, these high impulsive

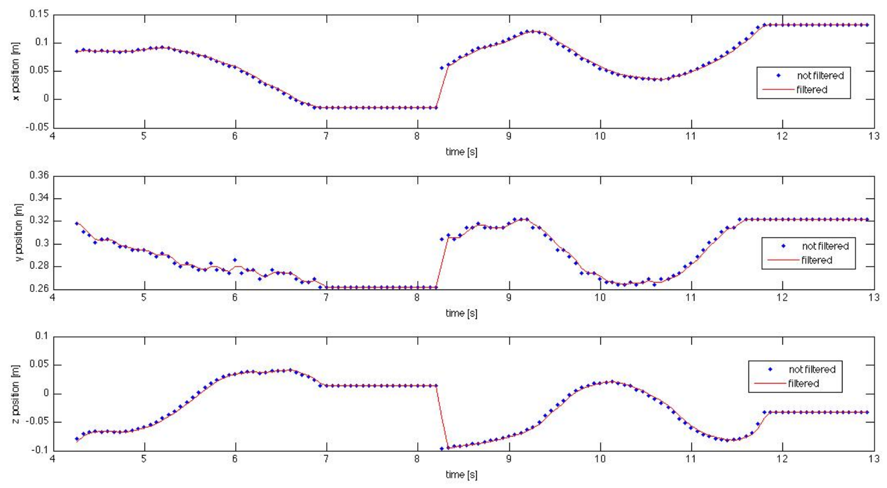


Figure 5.21: Position in the three axis using the "2 filter" (rotating).

changes can cause problems to the actuators. As you can see in the figure, the second order filter is not so effective when the position is found again, but it introduces a low time delay. In Figure 5.22, the signals filtered with the "10 filter" have been plotted with a red line. As you can see, the filtered signal is very smooth and continuous; this kind of filter is also very useful to remove the noise, expecially in the y-axis position; as already notices in the past experiment, the most noisy signal is the depth signal (y-axis), so it needs to be filtered properly. By the way, the time delay is too high and can cause problems in the actuation of the signal, since the actuated position has a high time delay.

Now, the position in the 3D space will be shown. In Figure 5.23 two 3D graphs are compared. The first one (on the left side) shows the position of the fingertip without applying a filter; as you can see, the movement is noisy. The noise is caused both by the vibrations on user hand movements and by the software fingertip position calculation. The second one (on the right) shows the 3D position after the "2 filter" has been applied. Some noise has been removed by using the filter, but the movement still remains a little bit noisy. In Figure 5.24 other two 3D graphs are shown. The 3D graph on the left side is the same a in Figure 5.23, while the graph on the right side represents the 3D position of the fingertip after the "10 filter" has been applied. As you can

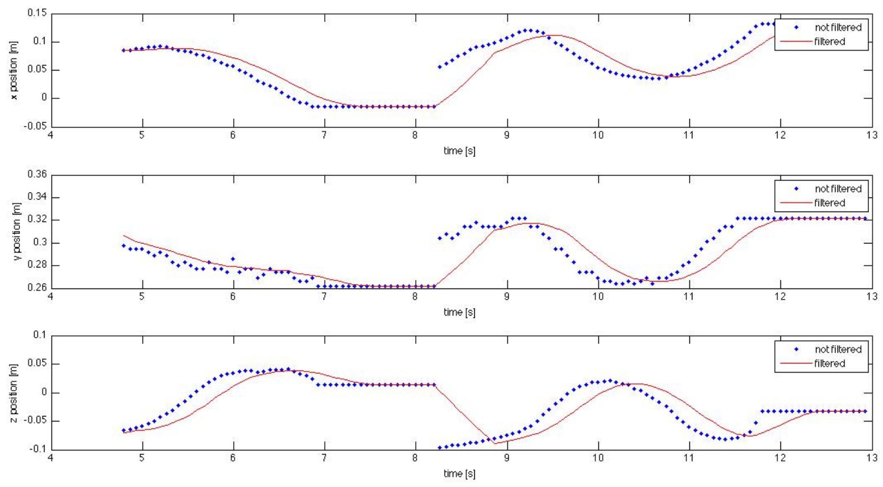


Figure 5.22: Position in the three axis using the “10 filter” (rotating).

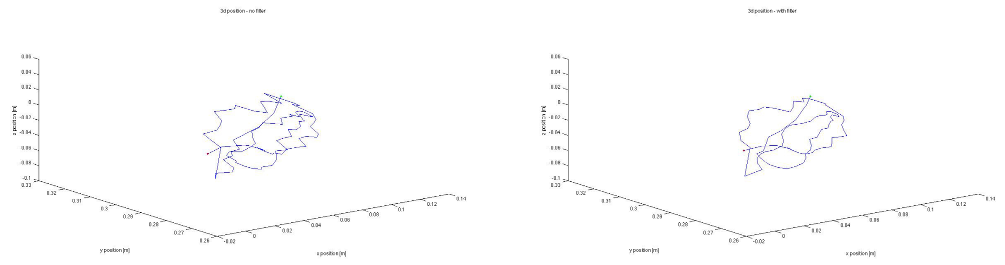


Figure 5.23: Fingertip rotating 3D position with “2 filter”.

see, the noise is filtered very well and the movement appears very smooth and continuous. In both graphs, the red point represents the starting point of the trajectory while the green point represents the ending point.

After the data has been shown, we can now summarize the pros and cons obtained by a filter implementation:

- as shown in many pictures, the signal in the 3D axis obtained by the program is noisy and needs to be filtered. This problem affects in particular the depth position (y-axis). Some noise needs to be removed in order to let the actuator work in a better way, without reaching any saturation limit. In fact, if the signal is too noisy, also the actuated

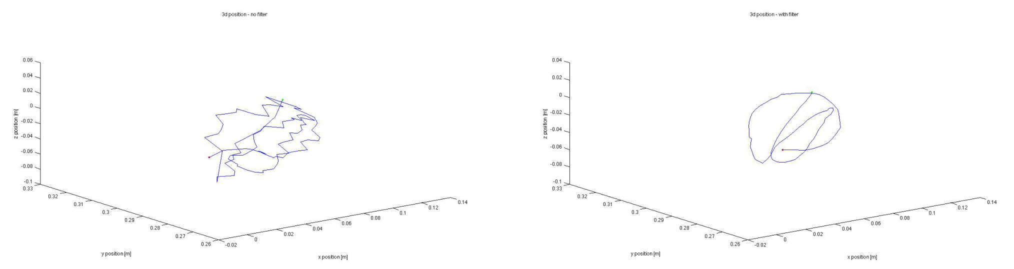


Figure 5.24: Fingertip rotating 3D position with "10 filter".

movement is noisy and this can cause many problems to the control and also noisy movements produce sound noise in the actuator.

- filters have proven to be useful in filtering high frequency noise. Some low pass filters have been implemented. These filters are basically moving average filters of a specified order; as the order becomes higher, the filter effectiveness becomes higher, too. Also, the filter is very important when the fingertip position is lost, because when the new position is retrieved the reference position is not a step signal (it becomes a ramp with a finite slope).
- increasing the order and the effectiveness of the filter is good, but this introduces also a phase delay (time delay) that can affect the position actuation in the system.

So, a filter is very useful, but it introduces also a time delay that must be limited. The decision of the filter is not easy and basically it is a compromise between filtering effectiveness and time delay.

Chapter 6

Pros/cons and problems encountered

In the previous chapters, the project has been widely explained. First of all, the purpose was to show the algorithms implemented in the software and also basic concepts necessary to understand the thesis. After that, a deep analysis of the program has been done: the program steps have been shown, from the image acquisition to the fingertip position filtering.

In this chapter, the purpose is to analyse pros and cons of the program, and also to talk about the problems encountered. At the end of the chapter, many future improvements will be proposed: future improvements can be implemented in the next projects and they're just some hints given by the user that can help to increase the system performances, increase the system reliability or to reduce the money expenses.

6.1 Pros and cons

In this subsection, pros and cons will be analysed. Some of them have already been mentioned in the previous chapters and so they will be explained for the second time. The field of human gesture recognition is wide and still many improvements must be implemented. In fact, the reliability of these systems is not so high since they refer to complex systems and many factors cannot be included in an analytical model and must be evaluated empirically. The pros of the proposed solution are the followings:

- the interface created is very simple. In fact, the only devices needed are two cameras and a personal computer running an open-source software. So, the interface is very compact and the space occupied by the system is very small. The total occupied space was less than one square meter.

- the interface does not require any contact between the user and the machine. This fact is very interesting, since interfaces without contact are more natural. In fact normally, when human beings operate to give some order, they do not use physical contact because the orders are given using hand movements or using the voice.
- the price of the interface is very small. The price is small because the system is composed by a computer and two cameras. The computer performances don't need to be too high, since the calculation time is not so high. For example, in the program, the frame rate is 15 fps for each camera. It means that the time required to grab the frames and analyse the data is less than 66 ms. In the project, a normal computer has been used and the price can be very low. In particular, the program does not require a lot of memory because the images are processed using a low resolution. The prices of the cameras depend on the quality of the optics and the quality of the sensors. A low distortion camera is required because optical distortion and chromatic aberrations can cause noise in the images or errors in the position calculation. The sofware used is all open-source software, so no additional costs are required: the operating system is Ubuntu and the libraries used are open-source libraries.

The cons of the proposed solution are the followings:

- the reliability of the system is not very high. In some cases, the fingertip tracked position is lost and so additional algorithms are needed to estimate the position of the fingertip during this deadlock. For some frames, the fingertip position cannot be calculated using the frames and so additional methods must be used. In this project, when the position is lost, the last reliable position is considered valid. By the way, it is also possible to use predictive systems based on neural networks. However, no predictive system has been used in this project; the effectiveness of a predictive system might be good if the fingertip position is lost for just some frames.

- the program is functioning only in some conditions. The created program can track only one fingertip at the same time; in particular, the tracked fingertip is the maximum reliability fingertip. In fact, the two biggest problems of the programs are to find the spacial correspondences in the left and right frames, but also to find the time correspondences. The second problem is difficult to be solved because it requires to find the correspondence between a fingertip in one frame and the same fingertip in the previous frame. If one of the frames have only one fingertip, the matching is very easy. However, if more fingertips are found in the present and past frames, the program must implement an algorithm to find the fingertip correspondeces in the two frames. In this project, the assumption that the user is using only one fingertip has been done; with this assumption, in fact, we just need to track one fingertip; in particular, the fingertip with the maximum reliability coefficient is considered the unique fingertip in the space. If more fingertips are present in the frames, the fingertip with the most similar color histogram is taken.

- sensitivity to the light. This program is sensible to the type of light used in the environment. The influence of the light has been already mentioned in the chapter related to the image acquisition. The light received by the sensor is the light reflected by each object; this light is produced by the light source in the environment. Example of light sources are the sunlight or a lamp; these kinds of sources have different electromagnetic spectra and so they introduce differences. The same object is seen in different ways by the camera depending on the kind of light source. Because of this reason, when the program starts, it's necessary to create a new histogram for both left and right frames. If the light conditions change in the environment, the reliability of the system is compromised and the color histograms must be recalculated.

- an output filter is necessary to filter the noise. The fingertip search algorithm introduces some noise expecially when calculating the depth of the fingertip (y-axis); in order to remove this noise, a discrete time filter is needed. The filter removes the noise but also it introduces a time delay in the signal, and this can be a problem when actuating the movement with an actuator.

- the position resolution is not so high. Expecially in the y-axis, the resolution is not high (3 mm in the example in the previous chapter). And also, the signal has some noise. In some applications, the actuation must be precise and so this resolution cannot be considered sufficient. Probably it is better to directly measure the depth of a point by using an infrared sensor.

Because of many cons in this project, many improvements must be implemented. In the next subsections, the encountered problems will be analysed and also some future improvements will be proposed.

6.2 Problems encountered

Many encountered problems have already been discussed in this chapter and in the previous. However, it might be interesting to underline some problems that have been encountered during the program operation. Pros and cons of this solution have already been reported, so the purpose of this subsection is just to remark some important points.

- one of the biggest limits of this program is that the program uses the contour information to search the fingertips. The method used to find the fingertips is based on the curvature of the hand contour. In particular, the assumption is that fingertips are located only where the contour has a big curvature. However, this assumption is not always true. In fact, as you can see in Figure 6.1, in this case the fingertips are not high curvature points; fingertips are not found and so the fingertip position is lost and it takes time to recover the fingertip position. In this case, the only way to find out the fingertip position is to use also the edge informations that can be retrieved by using an edge searching algorithm (for exampl the Canny edge detection algorithm). However, by using additional algorithms, the program might become very complex and the program operation could become difficult to be predicted. This problem can be solved by using a distance sensor, for example an infrared distance sensor. In fact, it has been noticed that normally fingertips are located at minimum depth positions. The depth distance of a point can be both measured (using an infrared sensor) or estimated (for example, in this thesis): directly measuring the distance is better because less approximations are introduced and also the resolution is higher.

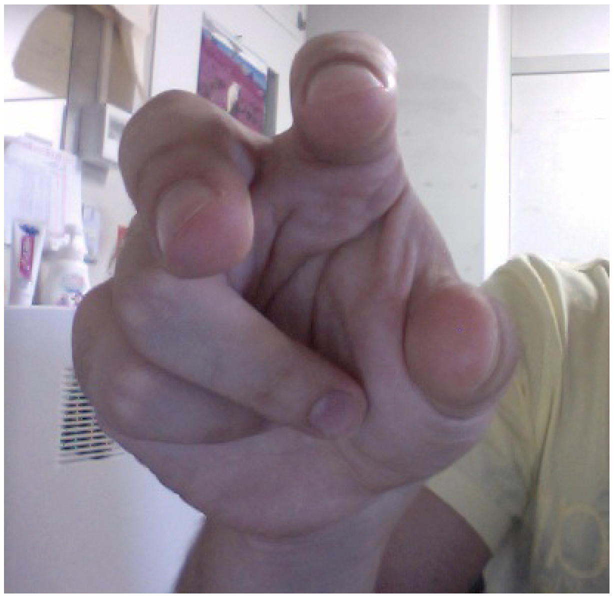


Figure 6.1: Fingertip occlusion.

- the fingertip position can be lost due to occlusion, noise or other causes. In this case, the position must be estimated or considered constant. In this project, the position is mantained constant if the fingertip position is lost. The loss of fingertip tracking can cause many problems to the actuator. These problems have already been widely discussed in this chapter and in the previous.

- an other problem that has been noticed in the program is caused by artificial light, i.e. by electrical lamps. In fact, electrical supply systems normally are using alternated current at 50 Hz or 60 Hz (depending on the countries). The light power produced by the lamp is not constant, but it changes depending on the time. For example, in a gas-discharge lamp in a 50 Hz electrical system, the discharge happens 100 times per second and the light flux is periodic but not constant. In the cameras, frames are grabbed 15 times per seconds. Because the two frequencies are not multiples, sometimes flicker phenomena can be observed in the grabbed frames when using this kind of lamps. By the way, this problem can be avoided by using other kind of lamps or lamps with high frequency power supply. This problem has not been noticed at daylight, because the daylight flux is constant.

The first two problems can be solved by adding some improvements; these improvements will be discussed in the next subsection. The third problem can be easily solved by using a controlled constant lighting source.

Chapter 7

Conclusion and future improvements

7.1 Conclusion

The purpose of this thesis was to deeply explain the contents of the research. In the technology field, many kinds of interfaces have been developed before; we can divide them in two big categories: interfaces requiring a physical contact and interfaces which don't require a physical concact. In this research, stereo vision has been used to track the position of a hand fingertip moving in a three dimensional space.

In Chapter 3, the experimental setup has been shown, in order to explain the structure of the system. The system is composed by two cameras connected to a personal computer using the Firewire bus. Also, basic details on the software have been given to the reader.

In Chapter 4, preliminary concepts have been explained. In fact, to easily understand the thesis, a basic knowledge of electronics, computer science, image and signal processing is required. In this chapter, many basic concepts have been given to the reader dividing them in two categories: explainations regarding image acquisition and filtering; then, the image processing algorithms used in the thesis have been explained. The purpose of this chapter was to explain to the user the processes used by the program to calculate the fingertip position in the three dimensional space.

In Chapter 5, the structure of the program has been deeply analysed. The purpose was to show the work which has been done by the author and explain the program operation. This chapter was divided in two parts: the first one explains the program operation from the image acquisition to the fingertip position calculation and also the graphical user interface; the second one is about the results. The results of the program operations have been reported and commented to show the program performances and limits; in order to remove the noise, additional discrete time filters have been used and their effectiveness has been discussed. In the project, some innovative techniques have been implemented by the author. For example, a smoothing filter has been used to filter the hand contour: this filter is very simple and the computation cost is very low, but it has proven to be very useful. Also, two kinds of noisy fingertips filtering have been used: the first one has been called "spacial filtering" and it uses the informations of the left and right frames images captured at the same time; the second ("time filtering") uses the informations of the fingertips in two frames taken at consecutive times to remove the residual noisy fingertips.

In Chapter 6, some considerations on the research have been done. First of all, pros and cons of the proposed solution have been analysed: in fact, interfaces based on stereo vision are very interesting and useful but they still have many problems and limits for many reasons. Then, some problems encountered have been reported. The final part of this chapter is related to possible future improvements that can be applied to the project; more specifically, future improvements can be both additional devices or more effective algorithms.

One of the targets of the thesis was to be understandable by many people. Because of this purpose, not only the results of the research have been reported, but also some introductory concepts have been given to the reader in order to make the thesis be easily understandable also by people that are not familiar with image acquisition and image processing.

7.2 Future improvements

Many improvements can be done to this project and some of them will be explained in this subsection. The possible improvements are the following:

- adding a distance sensor. The distance of a point in the space can be measured or estimated, depending on the system. In this project, the 3D position of each point has been estimated using triangulation.

Figure 7.1: Microsoft Kinect.

This method is inexpensive as it requires only two color cameras to be implemented; each object is searched in both left and right frames, and when the pixel position is found in both left and right frames, it is possible to calculate the 3D position. This method is inexpensive but the reliability and the resolution are not so high, and also the noise can be high. In the last chapter it has been shown that the depth distance (y-axis) is noisy: this noise is caused expecially by the fingertip distance calculation software. In order to reduce this noise, the best way is to integrate the camera system with a distance sensor. Nowadays, distance sensors based on infrared technology are common and they are also used in the game industry because of their relatively low price. In Figure 7.1 you can see the Microsoft Kinect system, an example of combined sensor (RGB color sensor and infrared depth sensor) used for gaming. In Figure 7.2 the Kinect block diagram is shown: there are both a RGB sensor and an infrared sensor. Combining them it is possible to calculate the 3D position of each point in the space. In particular, the depth sensor is very important, because it permits to get a 640x480 pixel image with a depth resolution of 11 bit (2048 discrete values). This method can be very useful because the position of an object is obtained with a higher accuracy than estimating it with a triangulation algorithm.

- evaluate the distance. In the thesis, to identify the fingertips, the contour curvature has been used. However, an other method could use the distance to evaluate the fingertip positions. An idea could be to identifiy the minimum distance points areas and consider them as fingertips. As you can see in Figure 6.1, the fingertips are located in correspon-

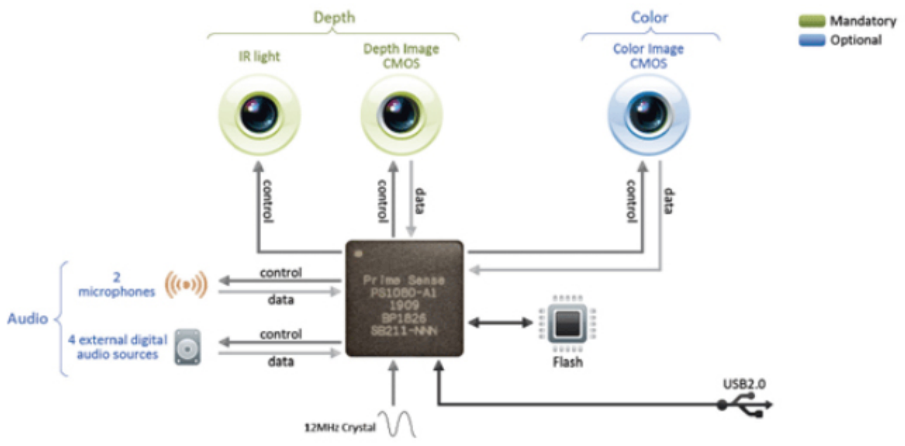


Figure 7.2: Kinect block diagram.

dence of minimum distance points. By using directly a distance sensor such as the Kinect, this target can be achieved with a higher accuracy.

- tracking more than one fingertip. In this thesis, only one fingertip has been tracked: the maximum reliability fingertip. However, in the reality, normally more fingertips are used to give some orders and more than one position needs to be retrieved. This is a big problem since it requires to create an algorithm to identify the correspondences of each fingers in two consecutive frames. If the fingertips are more than one, it could be difficult to solve the problem and a reliable algorithm must be implemented.

- adding a predictive method to calculate the position of the loss fingertips. When the fingertip position is lost, the last position is mantained constant. However, in this deadlock time, the fingertip is still moving in the space and it might be useful to estimate the position by evaluating the last fingertip position. This can be done maybe by using a neural network or other predictive methods.

The possible improvements listed are just some examples of future improvements that can be added to the software. The targets are increasing the realibility and the accuracy but also reducing the time delay and the noise. And of course the cost of the system must remain relatively low.

Bibliography

[1] J. Rehg, T. Kanade. "DigitEyes: Vision-Based Human Hand-Tracking". School of Computer Science Technical Report CMU-CS-93-220, Carnegie Mellon University, December 1993.

[2] Feng-Sheng Chen, Chih-Ming Fu and Chung-Lin Huang, "Hand gesture recognition using a real-time tracking method and hidden markov models", Image and Vision Computing, Vol. 21, No. 8, pp. 745-758, 2003.

[3] K. Oka, Y. Sato, and K. Koike, "Real-time fingertip tracking and gesture recognition", Computer Graphics and Applications, IEEE, Vol. 22, No. 6, pp. 64-71, 11-12-2002.

[4] Rung-Huei Liang and Ming Ouhyoung, "A real-time continuous gesture recognition system for sign language", Automatic and Gesture Recognition, 1998, Proceedings. Third IEEE International Conference on, pp. 558-567, April 1998.

[5] Anthony Isaac, Tomoaki Yoshikai, Hiroaki Yaguchi, Kei Okada, Masayuki Inaba, "Depth-based scope switching and image skeletonization for gesture recognition", Proceeding of the 2011 JSME Conference on Robotics and Mechatronics, Okayama, Japan, May 26-28 2011.

[6] Jennings, C., "Robust finger tracking with multiple cameras", University of British Columbia Vancouver, B.C, Canada.

[7] M. Jones, J. Rehg. "Statistical color models with application to skin detection". In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1999. Vol. 1, pp. 274-280.

[8] Jernej Mrovlje and Damir Vrančić, “Distance measuring based on stereoscopic pictures”, 9th International PhD Workshop on Systems and Control: Young Generation Viewpoint 1. - 3. October 2008, Izola, Slovenia.

[9] German K. M. A. “3D Reconstruction Algorithm Combining Shape-From-Silhouette with Stereo”. IEEE Transactions on PAMI. Vol.11, 2003:1265-1278.

[10] B. Stenger, P. R. S. Mendonça, R. Cipolla, "Model-Based 3D Tracking of an Articulated Hand," cvpr, vol. 2, pp.310, 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’01) - Volume 2, 2001.

[11] S. Malik, “Real-Time Hand Tracking and Finger Tracking for Interaction”. CSC2503F Project Report, 2003.Rafael C. Gonzales, Richard E. Woods, “Digital image processing”, Prentice Hall, 2008.

[12] Alan C. Bovik., “The essencial guide to image processing”, Elsevier, 2009.

[13] Mark S. Nixon, Alberto S. Aguado, “Feature extraction and image processing”, 1st Ed. Newnes, 2002.

[14] John C. Russ, “The image processing handbook”, 4th Ed. CRC Press, 2002.

[15] Wilhelm Burger, Mark J. Burge, “Principles of Digital Image Processing : Fundamental Techniques”, Springer-Verlag London, 2009.

[16] Junichi Nakamura, “Image sensors and signal processing for digital still cameras”, Taylor & Francis, 2006.

[17] Wilhelm Burger, Mark J. Burge, “Principles of Digital Image Processing : Core Algorithms”, Springer-Verlag London, 2009.

[18] Wilhelm Burger, Mark J. Burge, “Principles of Digital Image Processing : Fundamental Techniques”, Springer-Verlag London, 2009.

[19] E. Trucco, A. Verri, “Introductory Techniques for 3D Computer Vision”, Prentice-Hall, 1998.

[20] Bradski, G. and Kaehler, A., “Learning OpenCV: Computer Vision with the OpenCV Library”, 1st Ed. Sebastopol: O’Reilly, 2008.

[21] Intel Corporation, “Open Source Computer Vision Library: Reference Manual”, 1999-2001.

[22] Robert Whitrow, “OpenGL Graphics Through Applications”, Springer-Verlag London Limited, 2008.

[23] Shalini Govil-Pai, “Principles of Computer Graphics : Theory and Practice Using OpenGL and Maya”, Springer Science+Business Media, Inc., 2005.

[24] D’antona Gabriele, Ferrero Alessandro, “Digital signal processing for measurement systems”, Springer-Verlag, 2006.

Thanks

First of all, I would like to thank my parents because, with their own life, they have taught me the principle of respect to each other and the importance of love, and they also gave me the possibility to study engineering. In particular, I learned from my mom that loving someone is not only joy and pleasure but also sacrifice for the family and for work; I learned from my father the importance of being romantic, and also that even in disability life is still wonderful. After that, I really thank Paolo, my elder brother, because he opened the way in life; and his girlfriend, Angela: thank you for making me smile and for being such a good friend.

Many thanks to my friends in Buttapietra and near places (Marco, Rachele, Emma, Eugenia, Anna, Marina and many others), to the "Verona mountains" friends (Nadia, Giulia, Elena, Ester and families), to my high school classmates (Alessio, Ale, Cesco, Mattia and the others). I thank also Federica Canello and Federica Berardo because they taught me the importance of loving someone. Also, many thanks to my "milanese" friends at the Politecnico di Milano and at the Collegio di Milano (in particular to Chiara, Francesca and Gaia); thanks to Jackie, too, for all the good time we spent toghether.

Also, thanks to "zii" Teresa and Mario: they helped me to grow up and they have threated me like their own son. Thanks to my grandparents, uncles and cousins, especially to my uncles Luisa and Roberto: they have educated me since my birth and they really helped me in the past five years of studies in Milano.

I would like to thank also all my teachers since childhood. In particular, those of them who let me know math, physics and science. At high school, Vito Garieri and Paolo Gini, for being respectively my math and physics professors. At the Politecnico di Milano: Maria Stefania Carmeli, for assisting me when writing this thesis; moreover, Cesare Mario Arturi, Mario Ubaldini, Antonino Di Gerlando, Andrea Silvestri and Sergio Pignari: they haven't been only good professors, but they also integrated their lessons with useful advices about work and life. At Tokyo Institute of Technology, thanks to professors: Toshiharu Kagawa, Kenji Kawashima and to the assistant professor Kotaro Tadano; they have been my supervisors when writing my thesis in their laboratory.

Many thanks to all the people that I have met in Japan, coming from all over the world. They contributed to make this experience unique and unforgettable. From Japanese people, I tried to learn their ability of remaining calm and quiet even in critical situations. They also taught me the importance of community in their society, that contrasts so strongly with the individualism of western culture. Thanks also to my friends in the Komaba International House dormitory (expecially people from Korea and from Thailand), to my mates and Japanese language professors at Tokyo Institute of Technology and to everyone that introduced me to Tokyo lifestyle and Japanese culture and language (expecially Miki).

All these people taught me the principles of love, friendship, respect to each other and also the importance of studying and the pursuit of knowledge.

At last, I want to thank God for this life that has been given to me, for all the interesting people that crossed my line of life and for all the beautiful things that surround me in the Universe. I would like to mention some of them: the starlights in the sky, the sound of the waves in the sea, the smell of flowers, the taste of cherries and the softness of the lips of a woman. Experiencing all these feelings, I realise how wonderful the human being is with his infinite complexity, but also how infinitely small we really are in the Universe.