UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Introduzione all'utilizzo del sensore Kinect in ambiente UNIX

Elaborato di esame di:

Camilla Panni, Alessandro Musatti

Consegnato il: **13 giugno 2013**

Sommario

Il lavoro svolto ha il fine di introdurre l'utilizzo del sensore Kinect in ambiente UNIX, e di fornire una libreria scritta in linguaggio C che fornisca supporto all'utilizzo della Kinect come sensore di scansione laser per il rilevamento di ostacoli e utilizzabile da processi esterni, e di un relativo programma di esempio. Si provveduto in prima battuta all'installazione della libreria libfreenect, utilizzata come punto di partenza in unione con la libreria OpenCV per analizzare la tipologia di dati e prendere coscienza del funzionamento del sensore. Come passo successivo si sono attuate delle tecniche di image processing e la calibrazione del sensore mediante la libreria ROS per migliorare la qualità delle misurazioni ottenute dal sensore. Infine si è realizzata un'applicazione che supporta la comunicazione inter-processo attraverso la memoria condivisa, in grado di fornire i dati di scansione ad altri processi.

1. Introduzione

Questo elaborato mira all'introduzione all'utilizzo del sensore Kinect in ambiente Unix e alla realizzazione di una libreria per la manipolazione dei dati ricevuti in output dal sensore con il fine di ottenere delle funzioni utilizzabili in collaborazione tra loro da processi esterni.

1.1. Descrizione del sensore

Il Kinect, inizialmente conosciuto con il nome di Project Natal, è un dispositivo di input per la console di gioco di casa Microsoft, l'Xbox 360. Kinect è stato annunciato al pubblico il 10 giugno 2009 durante la conferenza stampa della Microsoft all'E3 2009 con il nome Project Natal, poi rinominato Kinect alla presentazione ufficiale all'E3 2010.

L'ingresso nell'attuale mercato video-ludico di questo apparecchio ha portato ad una rivoluzione nel modo di giocare dato che questo strumento permette di comandare azioni di gioco senza l'utilizzo di joypad o pulsanti ma semplicemente con l'utilizzo del proprio corpo e, a differenza del controller Wii della Nintendo o del Motion Controller Sony, questo sensore non ha bisogno di alcun elemento da collegare al corpo per decodificare i movimenti effettuati dal giocatore. Sebbene in origine pensata per Xbox 360, Microsoft prevede di rendere nel prossimo futuro disponibile l'uso della periferica a tutti i PC dotati di un sistema operativo Windows.

Fig. 1 - Il sensore Kinect

Il Kinect è stato sviluppato per la Microsoft dall'azienda israeliana Primesense il cui obiettivo fu appunto la creazione di un controllore del corpo umano privo di sensori da applicare al soggetto. Fin dal giorno della sua uscita sul mercato (4 novembre 2010) il Kinect è entrato a far parte della realtà di sviluppatori universitari o freelance i quali, con tecniche di reverse engineering hanno iniziato a studiare questo dispositivo. La Microsoft, la quale inizialmente aveva dichiarato di essere contraria all'utilizzo del Kinect all'infuori dell'ambiente di gioco, intuì che il prodotto aveva grandi potenzialità anche con differenti dispositivi e in data 10 novembre 2010 fece uscire i primi driver che permisero l'utilizzo della

telecamera RGB e del sensore di profondità. Recentemente la Microsoft ha dichiarato che il Kinect sarà parte integrante del nuovo sistema operativo Windows 8.

1.1.1. Descrizione Hardware del sensore

In questa sezione è fornito un accenno all'hardware presente nel dispositivo, al fine di identificarne uno schema di funzionamento basato sulla conoscenza e connessione delle sue componenti materiali prima che su quelle concettuali e algortmiche fornite nel seguito. L'hardware di Kinect si basa su tecnologie di 3DV**,** una compagnia che Microsoft ha prima finanziato e poi acquisito nel 2009, nonché sul lavoro dell'israeliana PrimeSense**,** che ha poi dato in licenza la tecnologia a Microsoft. Il 13 maggio 2010 è stato pubblicato negli Stati Uniti da PrimeSense un brevetto dal titolo "Depth Mapping using Projected Patterns" che spiega esattamente la tecnologia implementata in Kinect.

Il dispositivo dopo un disassemblaggio si presenta nel modo mostrato in Figura 2:

Fig. 2 - Le parti del sensore

L'apertura del dispositivo mostra la presenza di:

- un array di microfoni orientati verso il basso (3 sul lato destro e uno sul lato sinistro);
- tre apparati ottici utilizzati per il riconoscimento visuale del corpo in movimento;
- una ventola per la dissipazione del calore;
- 64MB di memoria flash DDR2;
- un accelerometro Kionix KXSD9 a tre assi;
- Prime Sense PS1080--A2, il chip che rappresenta il cuore della tecnologia di Kinect.

Più in dettaglio il corredo di apparati ottici di Kinect si compone di una telecamera RGB e un sensore di profondità a raggi infrarossi. Tale sensore è composto da un proiettore a infrarossi e da una telecamera sensibile alla stessa banda, che viene utilizzata per leggere quanto rilevato dai raggi infrarossi. La telecamera RGB ha una risoluzione di 640 × 480 pixel, mentre quella a infrarossi usa una matrice di 320 × 240 pixel. L'array di microfoni è usato dal sistema per la calibrazione dell'ambiente in cui ci si trova, mediante l'analisi della riflessione del suono sulle pareti e sull'arredamento. In tal modo il rumore di fondo e i suoni del gioco vengono eliminati ed è possibile riconoscere correttamente i comandi vocali. La barra del Kinect è motorizzata intorno l'asse orizzontale e segue i movimenti dei giocatori, orientandosi nella posizione migliore per il riconoscimento dei movimenti.

Mediante studi sperimentali, è stato determinato che il sensore di Kinect processa il flusso video ad un frame rate di 30 Hz. Lo stream RGB usa una risoluzione VGA ad 8 bit di profondità con dimensione di 640x480 pixels con un filtro di Bayer per il colore, mentre il sensore monocromatico di profondità

genera un flusso dati a risoluzione VGA(640 x 480 pixels) con 11 bit di profondità, quindi con 2048 valori possibili.

Il sensore di Kinect ha un range di utilizzo da un minimo di 0.4 ad un massimo di 4.0 metri. Esso ha un campo di vista angolare di 57° orizzontalmente e 43° verticalmente, mentre il fulcro motorizzato é capace di uno spostamento di 27° gradi verso l'alto o verso il basso, controllabile via software. Il campo orizzontale ad una distanza di 0.8 m è all'incirca 87 cm, mentre quello verticale circa 63 cm, risultando quindi una risoluzione di 1.3 mm per pixel. La risoluzione spaziale x=y a 2 metri di distanza dal sensore é di 3 mm, mentre la risoluzione di profondità z sempre a due metri di distanza è di 1 cm.

Come già accennato, la periferica permette all'utente di interagire con la console senza l'uso di alcun controller da impugnare, ma solo attraverso i movimenti del corpo, i comandi vocali o attraverso gli oggetti presenti nell'ambiente. Microsoft dichiara che Kinect può seguire i movimenti di fino a quattro giocatori, sia in piedi che seduti.

Visti i componenti hardware, uno schema di funzionamento del dispositivo potrebbe essere il seguente presentato in Figura 3:

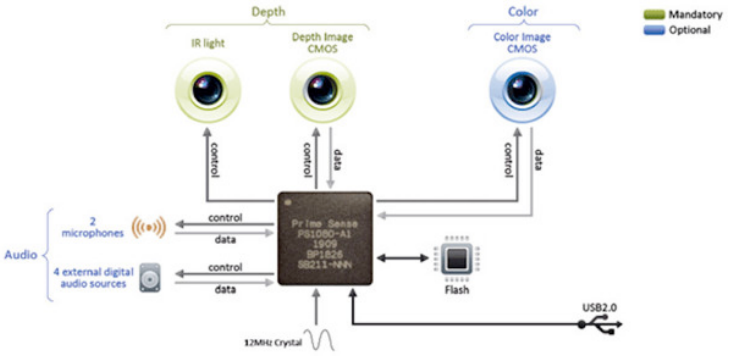

Fig. 3 - Schema di funzionamento

nel quale viene evidenziato come il chip PS1080--A2 di PrimeSense sovraintenda tutta la procedura di analisi della scena controllando adeguatamente gli apparati ottici e audio al fine di raccogliere le informazioni di cui necessita. A puro scopo informativo di seguito un elenco dei vari chip presenti all'interno di Kinect:

- Wolfson Microelectronics WM8737G – Stereo ADC con microfono preamp;
- Fairchild Semiconductor FDS8984 – N--Channel PowerTrench MOSFET;
- NEC uPD720114 – USB 2.0 hub controller;
- H1026567 XBOX1001 X851716--005 GEPP;
- Marvell AP102 – SoC con interfaccia Camera controller;
- Hynix H5PS5162FF 512 megabit DDR2 SDRAM;
- Analog Devices AD8694 ; Quad, Low Cost, Low Noise, CMOS Rail--to--Rail Output Operational Amplifier ;
- TI ADS7830I – 8--Bit, 8--Channel Sampling A/D Converter with I2C Interface;
- Allegro Microsystems A3906 – Low Voltage Stepper and Single/Dual DC Motor Driver;
- ST Microelectronics M29W800DB – 8 Mbit (1Mb x8 or 512Kb x16) NV Flash Memory;
- PrimeSense PS1080--A2 – SoC image sensor processor;
- TI TAS1020B USB audio controller front and center.

Completa la lista dell'hardware del dispositivo un accelerometro Kionix MEMS KXSD9, utilizzato per controllare l'inclinazione e stabilizzare l'immagine.

1.2. Funzionamento del sensore

In questa parte vengono dati dei cenni su quali sono le tecniche, utilizzate dal sensore Kinect, alla base del riconoscimento dei movimenti del giocatore per poi riuscire a renderlo controller. Principalmente si dividono in due passi principali: viene costruita una mappa di profondità utilizzando l'analisi di luce strutturata creata mediante l'emettitore di infrarossi e infine viene dedotta la posizione utilizzando algoritmi di tracking implementati nel software sviluppato da Microsoft. Verrà approfondito il concetto di disparity map in quanto di maggiore interesse rispetto allo sviluppo del progetto presentato.

1.2.1. Disparity map

Consideriamo due immagini di una stessa scena prodotte mediante un sistema di stereovisione. Una mappa di disparità può essere vista come una matrice M di numeri interi di dimensione WxH, dove W ed H sono rispettivamente l'ampiezza e l'altezza dell'immagine.

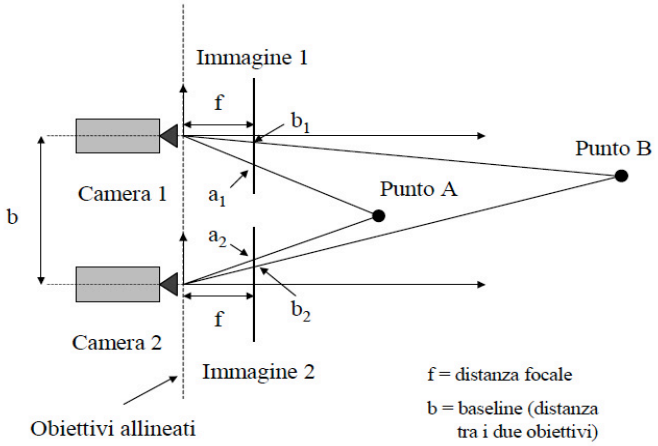

Fig. 4 - Sistema di telecamere

Ogni intero d presente nelle celle della matrice rappresenta la distanza, espressa in pixel, tra il pixel di riferimento p, preso dalla prima immagine, ed il corrispondente pixel p′, appartenente alla seconda immagine, nel momento in cui le due immagini vengono sovrapposte. Più è alto il valore della disparità d, più il punto tridimensionale, rappresentato univocamente dalla coppia di pixel p e p′, si trova vicino alle due camere. Di conseguenza la disparità d(A) calcolata per il punto A deve essere maggiore rispetto alla disparità d(B) del punto B, poiché sicuramente A si trova più vicino alle due camere rispetto a B. Quanto appena affermato però è vero solo a condizione che gli obiettivi delle due camere siano allineati e lievemente distanziati come illustrato in Figura 4. Definiamo ora come a1 e a2 i punti proiezioni di A rispettivamente sulla prima e sulla seconda immagine; analogamente definiamo b1 e b2 per il punto B. Assunto inoltre che tutti i punti a1, a2, b1 e b2 abbiano lo stesso valore di ordinata, allora tali punti giacciono tutti sulla medesima linea orizzontale.

Calcoliamo dunque i valori di disparità per i punti A e B nel seguente modo:

disparità per il punto A: $d(A) = Xa1 – Xa2$

disparità per il punto B: $d(B) = Xb1 – Xb2$

notiamo che d(A) > d(B) essendo Xa1 >Xb1 e Xa2 < Xb2. E' importante sottolineare che sono da considerare soltanto i valori di ascissa dei quattro punti. in virtù dell'ipotesi di allineamento verticale, come illustrato in Figura 5:

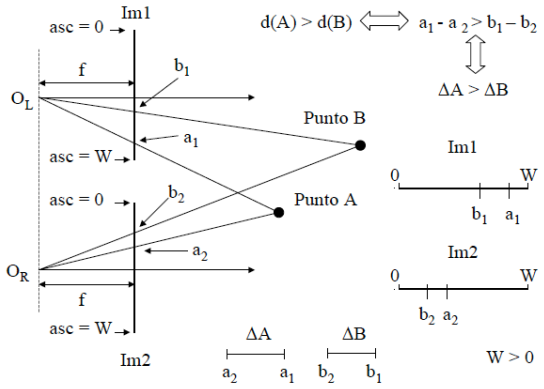

Fig. 5 - Schema calcolo disparità

se consideriamo le due immagini come matrici di pixels, i punti a1 e a2 possono essere rappresentati in base ai rispettivi valori di riga e di colonna. Il pixel a1 avrà coordinate riga-colonna (i, j) mentre a2 si troverà in posizione (i′, j′).

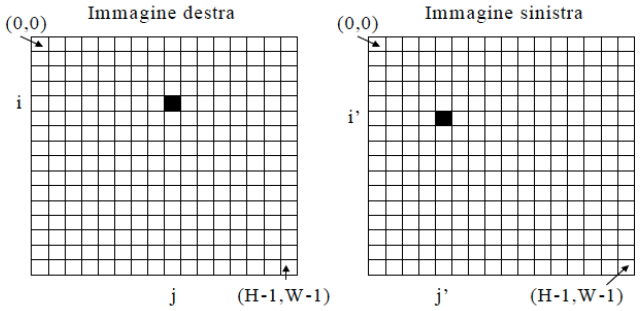

Fig. 6 - Rappresentazione in matrice di pixel

La disparità tra a1 e a2 altro non sarà che la distanza tra le locazioni (i, j) e (i′, j′) quantificata mediante un intero che indichi la differenza in pixel tra le due posizioni della coppia coniugata. Chiaramente, la disparità può essere calcolata solo per punti della scena che sono visibili in entrambe le immagini; un punto visibile in un'immagine ma non all'altra viene detto essere occluso. Una volta che la matrice M è stata completamente riempita, ovvero una volta che per ogni pixel nell'immagine di riferimento è stato trovato il corrispettivo nella seconda immagine ed è stata calcolata la disparità tra i due, abbiamo ottenuto una mappa della disparità della scena che stiamo analizzando. Associando l'informazione contenuta in M sotto forma di numeri interi, che vanno da zero a DMAX, dove DMAX è la massima disparità ammessa, ad una qualche scala di colori, possiamo ottenere una terza immagine che rappresenta gli oggetti della scena con differenti colori in base alla loro distanza dalle camere. Da quanto detto si può dunque dedurre che una mappa di disparità può essere usata come strumento per la valutazione tridimensionale dell'ambiente osservato.

1.3. Conversione coordinate di un punto 3D nello spazio reale

Le camere del sensore Kinect effettuano una trasformazione dei punti reali nello spazio 3D proiettandoli sul piano ottico formando un'immagine del modo reale sul CCD. Questa trasformazione è data dalla formula seguente:

$$\begin{bmatrix} u \\ v \\ Z \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Ogni punto con coordinate reali nello spazio X,Y,Z viene proiettato sul punto con coordinate u,v dell'immagine 2D. Il riferimento dei punti nello spazio ha origine nel centro ottico della camera e la distanza Z è fornita direttamente dalla depth camera come distanza fra il punto e il piano ottico della IR camera. La matrice di trasformazione è detta matrice intrinseca e i valori fx ed fy sono rispettivamente le distanze focali del sistema di lenti utilizzato nella camera, mentre cx e cy sono le coordinate in pixel del centro dell'immagine del sistema ottico rispetto all'origine posta nell'angolo in alto a sinistra.

Da questa trasformazione risulta possibile ricavare quella inversa che permette di trasformare i punti dell'immagine con coordinate u,v in punti reali dello spazio con coordinate X,Y,Z. Le unità di misura nelle quali sono espresse le coordinate reali dei punti dipendono dai valori utilizzati nella matrice intrinseca, relativamente a come si è effettuata la calibrazione della camera per determinare la matrice intrinseca (vedi sezione calibrazione-numero).

Risulta quindi immediata la trasformazione inversa mediante la seguente formula:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}^{-1} * \begin{bmatrix} u \\ v \\ Z \end{bmatrix}$$

Il sistema ottico del sensore Kinect è dotato di due telecamere differenti e dislocate in posizioni differenti del sensore, da questo ne risulta che le immagini fornite dalle due telecamere non possono coincidere. Per poter trasformare le coordinate u,v dei punti di una telecamera nelle coordiante s,t dei punti dell'altra telecamera è necessario effettuare la calibrazione estrinseca che permette di ricavare la matrice di rotazione e quella di traslazione.

Per convertire le coordinate di un punto u,v in s,t si utilizza la formula-numero:

$$\begin{bmatrix} s \\ t \\ w \end{bmatrix} = R * \begin{bmatrix} u \\ v \\ w \end{bmatrix} + T$$

dove R è la matrice di rotazione e T è il vettore di traslazione riferito alle due telecamere in questione. Si noti che la terza componente dei vettori nella trasformazione è posta a 1 in quanto non viene considerata per punti nello spazio 2D.

2. Il problema affrontato

Il problema affrontato in questo elaborato consta nell'utilizzo del sensore Kinect in ambienti UNIX sfruttando la libreria open-source libfreenect, con lo scopo di riprodurre le funzionalità di un sensore laser, in grado di fornire in uscita una serie di misurazioni sfruttabili da processi esterni.

2.1. Disallineamento tra l'RGB e la depth image

La RGB e la depth image risultano disallineate a causa di differenti fattori ad esempio della distanza tra la RGB camera e la IR camera, dalla rotazione degli assi ottici e infine dalle distorsioni introdotte dalle lenti; tutte questi determinano per ogni camera un campo visivo differente. La dimensione della depth image risulta diminuita a causa del minore campo visivo rispetto a quello della RGB camera.

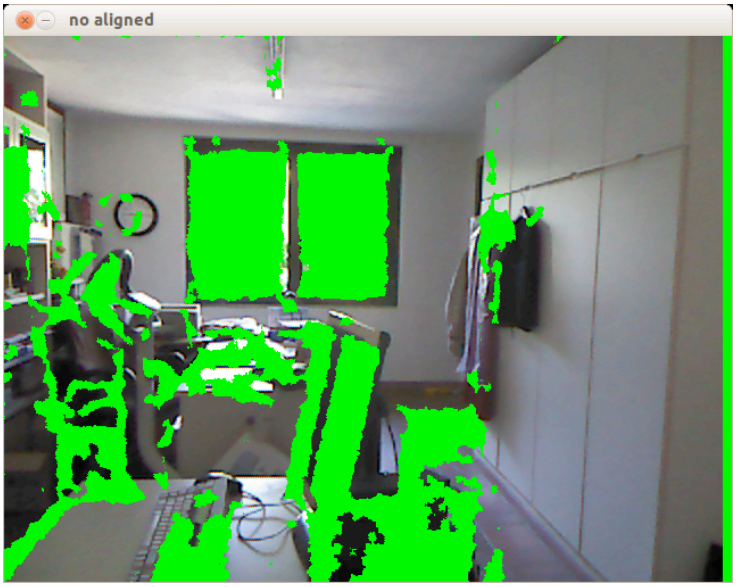

Fig. 7 - Depth image sovrapposta all'RGB

In Figura 7 si può notare il disallineamento sovrapponendo la depth image all'RGB, le zone dell'immagine clorate in verde rappresentano tutte le zone d'ombra in cui la distanza assegnata dal sensore ai punti è zero, in questo modo si mette in evidenza come non vi sia corrispondenza fra le zone d'ombra e i bordi degli oggetti, ad esempio si noti come l'ombra dell'orologio sullo sfondo non è allineata in modo corretto al bordo dell'orologio nell'RGB.

2.2. Il rumore della depth image

L'output della depth camera è affetto da un particolare tipo di rumore che fa oscillare il valore della distanza dei punti fra un valore valido e zero nel tempo. Questo tipo di fenomeno si traduce in una continua oscillazione del valore della distanza dei punti, soprattutto sui bordi degli oggetti visualizzati, impedendo una corretta individuazione dei contorni.

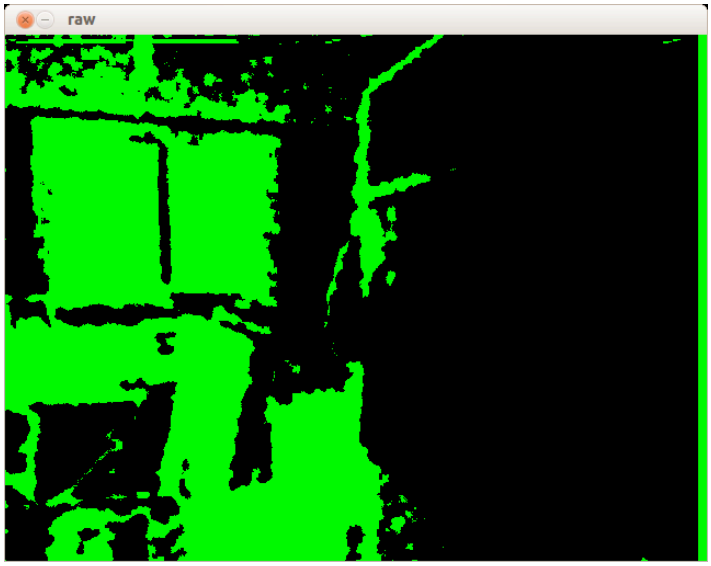

Fig. 8 - Depth image non filtrata

In Figura 8 si può notare l'effetto del rumore introdotto dalla depth camera, il colore verde rappresenta anche in questo tutti i punti con distanza nulla, in particolare i bordi degli oggetti.

2.3. Ombre nella depth image

A causa della distanza tra l'IR camera e l'illuminatore gli oggetti formano ombre nella depth image. Il sensore non è in grado di stimare la distanza per tali zone in ombra e pertanto il loro valore di profondità è settato a zero, la Figura 9 seguente illustra il principio per cui questo fenomeno accade:

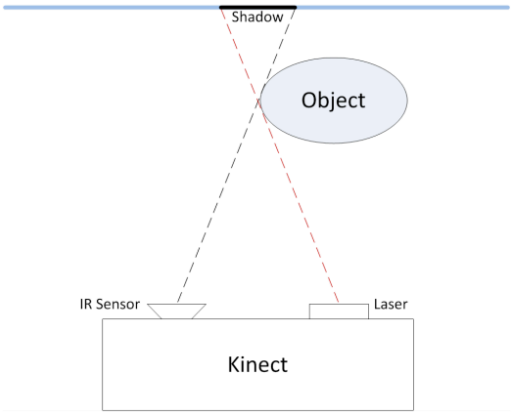

Fig. 9 - Fenomeno dell'ombra

Viene qui presentata un'immagine in Figura 10 che raffronta l'immagine RGB e la depth image in cui sono ben visibile sul lato sinistro degli oggetti le ombre prodotte dal fenomeno descritto in precedenza.

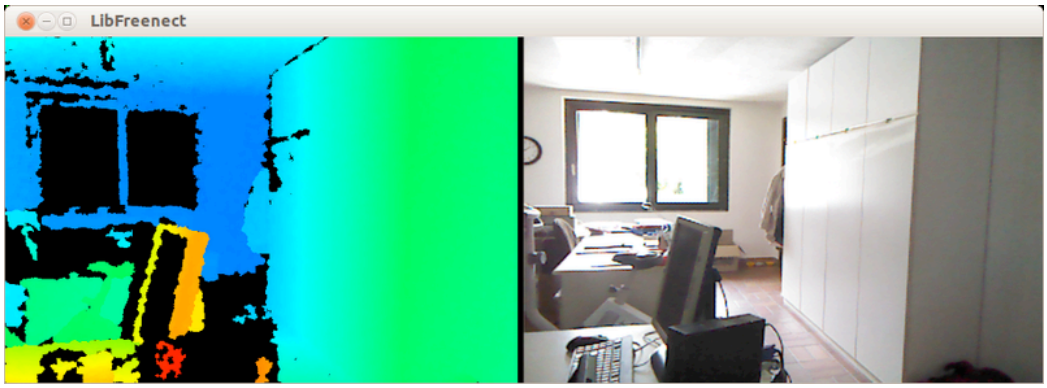

Fig. 10 - Depth image e RGB a confronto

Sulla sinistra è rappresenta la depth image mediante l'utilizzo di una scala di colori per segmentare i valori della distanza, il nero rappresenta il valore nullo di questa.

3. La soluzione adottata

3.1. Allineamento tra l'RGB e la depth image

La libreria libfreenect fornisce una serie di opzioni di elaborazione delle immagini originali provenienti dal sensore. Una di queste opzioni è relativa al formato di dati della depth image. Modificando il file relativo al wrapper opencv chiamato *libfreenect_cv.c* è possibile modificare la funzione freenect_sync_get_depth_cv() in modo che attraverso l'opzione FREENECT_DEPTH_REGISTERED fornita alla funzione del wrapper c_sync freenect_sync_get_depth(), sia possibile ottenere automaticamente una versione della depth image già correttamente allineata e adattata alla RGB image. Mediante questa opzione si ottiene una depth image allineata il cui valore della distanza dei punti è espresso in millimetri al posto del valore di profondità di default a 11 bit che non garantiva l'allineamento.

Un'altra possibile soluzione per l'allineamento è quella di effettuare la calibrazione estrinseca della depth camera e della RGB camera, in modo tale da rendere possibile una trasformazione dei punti corretta mediante le matrici di traslazione e rotazione delle camere.

Viene qui riportato in Figura 11 i risultati ottenuti dopo l'allineamento:

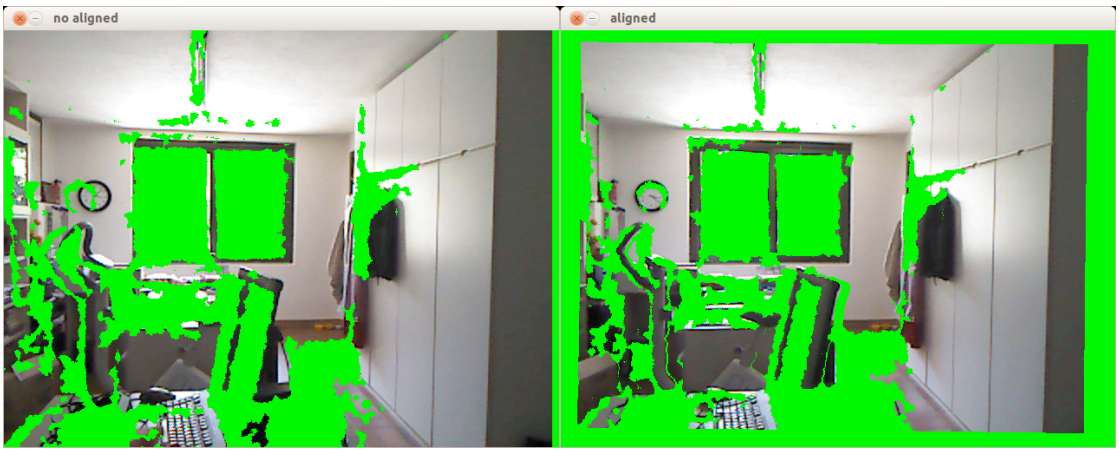

Fig. 11 - Esempio allineamento RGB e depth image

3.2. Processo di filtraggio della depth image

L'output della depth camera oltre che all'incertezza di misura, è affetto da un particolare tipo di rumore che fa oscillare nel tempo il valore di distanza dei punti, fra un valore ammissibile e zero. Questo tipo di fenomeno si traduce in un cambiamento nel valore della distanza dei punti, soprattutto sui bordi degli oggetti visualizzati. Per ovviare a questo fenomeno si è deciso di filtrare il valore della depth camera nel tempo, ogni punto viene classificato in base alla variazione del suo valore. Ad ogni punto sono associati due attributi specifici, un contatore sul numero di volte in cui il valore di distanza non è nullo, ed uno che mantiene l'ultimo valore valido. Ogni punto viene classificato a seconda del comportamento che assume il valore della distanza nel tempo, ovvero se il valore della distanza non è nullo il valore filtrato è congruente al valore in ingresso. Viceversa quando un punto inizia ad avere valori nulli, il filtro mantiene in uscita l'ultimo valore valido e decrementa il relativo contatore. Fino a che questo contatore non risulta sufficientemente negativo il filtro continua a mantenere in uscita l'ultimo valore valido disponibile e solamente quando si raggiunge una determinata soglia il valore in uscita viene posto a zero. In questo modo il filtro cerca di attenuare la continua variazione di alcuni punti, cercando di fornire il più possibile un valore ammissibile, senza però introdurre un'eccessiva inerzia ai cambiamenti di valore, relativi ad esempio allo spostamento del sensore.

All'immagine in uscita da questo primo processo di filtraggio è applicato un secondo filtro, che effettua una media spaziale dei valori. Il filtro spaziale media il valore corrente con il valore dei punti ad esso vicino, in questo modo si cerca di uniformare il valore delle distanze di punti lontani, che appaiono più sgranati a causa della perdita di risoluzione del sensore.

Viene presento un esempio dell'applicazione del processo di filtraggio nella Figura 12:

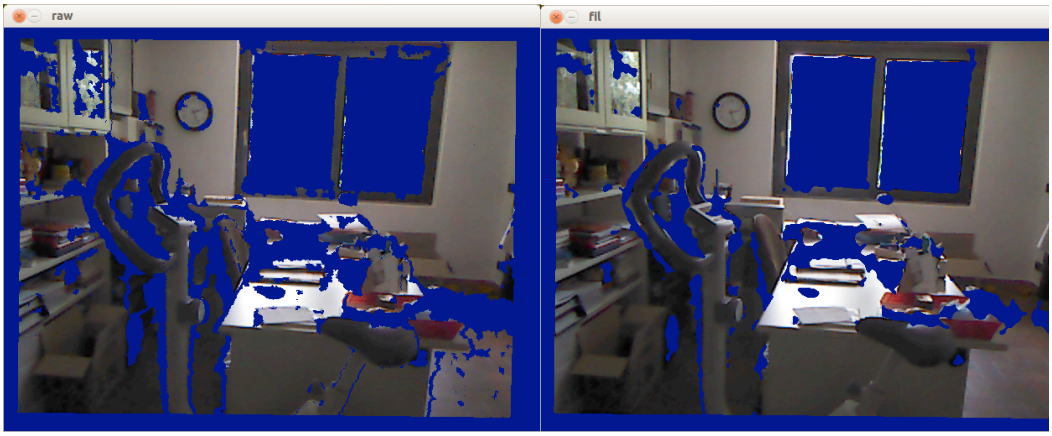

Fig. 12 - Esempio processo di filtraggio

3.3. Calcolo delle coordinate reali

La depth image fornisce il valore della distanza dei punti misurato dal sensore Kinect, inoltre per ottenere l'ascissa e l'ordinata dei punti è necessario calcolare la loro posizione tramite i dati ricavati dalla calibrazione della IR camera.

Di seguito sono riportate le formule utilizzate per il calcolo delle coordinate reali x,y,z dei punti:

$$x = \frac{(u - cx + \frac{W}{2}) \cdot z}{fx}$$
$$y = \frac{(v - cy + \frac{H}{2}) \cdot z}{fy}$$

dove u,v sono le coordinate dei punti della depth image in riferimento ad un piano cartesiano con origine nel centro dell'immagine, cx e cy sono le coordinate in pixel del centro dell'immagine reale fornita dalla IR camera, mentre fx e fy sono le lunghezze focali in pixel della IR image.

Il calcolo fornisce il valore in millimetri delle coordinate x e y di un punto con coordinate u e v, la distanza z è invece fornita direttamente dal sensore Kinect.

3.4. La libreria realizzata

3.4.1. scan.h

Il file scan.h contiene le definizioni dei parametri fondamentali all'utilizzo delle funzioni realizzate. In particolare contiene i *define* relativi ai valori per la depth camera:

- DF_X, DF_Y = lunghezza focale della IR camera espressa in pixel;
- DC_X, DC_Y = distorsione ottica della IR camera espressa in pixel.

Sono inoltre riportate le definizioni di due macro che risultano utili per convertire le coordinate dei punti rispetto al punto di origine dell'immagine.

LINC(px,py): restituisce l'indice dell'elemento dell'immagine che ha le coordinate px e py con l'origine degli assi nel centro dell'immagine.

Questa macro è utilizzata ogni volta che s'intende convertire le coordinate di un punto riferite ad un sistema di assi cartesiani centrato nel centro dell'immagine, per accedere al pixel relativo nel vettore puntato dal campo imageData della struttura Iplimage.

LINT(px, py): restituisce l'indice dell'elemento dell'immagine che ha le coordinate px e py con l'origine degli assi on top all'immagine.

Questa seconda macro è del tutto identica alla prima, con l'unica differenza nell'origine degli assi cartesiani che è posizionata centralmente e in alto nell'immagine.

Si noti che queste due macro possono essere utilizzate solamente per immagini con dimensioni definite da W e H, che nel caso specifico sono di 640 e 480 pixel.

In questo file sono anche riportate le strutture utilizzate per riferirsi ai punti:

- *point_3d* struttura di un punto 3d composta da 3 interi che rappresentano i valori delle coordinate x,y,z, di un punto nello spazio.
- *point_2d* struttura di un punto 2d composta da 2 interi che rappresentano i valori delle coordinate u,v di un punto in un'immagine.

Nel file sono inoltre presenti tre *define* che si riferiscono ai parametri relativi al shared memory segment[1]

[1] Vedi l'appendice A

- *SM_KEY* Si riferisce al valore di tipo *key_t* e rappresenta l'id del segmento condiviso di memoria.
- *SM_PERM* Si riferisce ai permessi di accesso assegnati al segmento di memoria condiviso, il valore specifico è 0644 (in ottale) ovvero: -rw-r--r--
- *SM_SIZE* La grandezza in byte del segmento, che in questo caso coincide alla grandezza di un vettore di W elementi di tipo point_3d.

3.4.2. scan.c

Il file scan.c contiene le funzioni sviluppate.

void **real_3d**(IplImage *depth, int px, int py, point_3d *pt3d)

La funzione real_3d() effettua la trasformazione inversa per ricavare la misura reale in millimetri dei punti nello spazio.

INPUT:

- IplImage * depth: puntatore dell'immagine di profondità, utilizzato nella funzione per ricavare il valore in millimetri della distanza dei punti dal piano focale del Kinect;
- int px: la coordinata y in pixel del punto da convertire, rispetto al centro dell'immagine;
- int py: la coordinata x in pixel del punto da convertire, rispetto al centro dell'immagine .

OUTPUT: point_3d *pt3d Questo puntatore si riferisce alla struttura di tipo point_3d che deve essere creata e passata alla funzione, dove vengono scritti i valori delle coordinate x,y,z espressi in millimetri del punto px,py.

void **remove_noise**(IplImage *in, IplImage *out)

La funzione remove_noise() effettua il processo di filtraggio, presentato nel paragrafo 3.2, sull'immagine in ingresso e restituisce l'immagine filtrata in uscita. Questa funzione non necessita di un tempo di transitorio per generare l'uscita, ma introduce un lieve sfasamento temporale tra input e output dovuto all'elaborazione delle immagini. Si noti che la funzione filtra l'intra immagine di WxH pixel.

INPUT: IplImage *in, puntatore all'immagine di profondità in ingresso al filtro

OUTPUT: Iplimage *out, puntatore all'immagine di uscita dal filtro

void **draw_shadow**(IplImage *depth, IplImage *rgb)

Questa funzione visualizza in modalità grafica tutti i punti della depth image con distanza pari a zero. Si noti che nel caso si intenda sovrapporre i punti non validi della depth image sui rispettivi punti dell'immagine proveniente dalla RGB camera, è demandato all'utente il compito di fornire una depth image allineata alla RGB image.

INPUT: IplImage *depth, puntatore alla depth image

OUTPUT: IplImage *rgb, puntatore all'immagine in output dove deve essere disegnata l'ombra

void **scan_2d(**IplImage *depth, int py, point_3d *line)

Questa funzione effettua la scansione 2d della depth image sul piano con ordinata *py*. Il risultato di questa scansione consiste in un vettore di W elementi di tipo point_3d dove sono contenuti i valori in millimetri delle coordinate reale dei punti. Si noti che il valore delle ordinate dei punti del vettore è il medesimo per tutti gli elementi.

INPUT:

- IplImage *depth: puntatore alla depth image
- int py: valore in pixel dell'ordinata del piano da considerare per effettuare la scansione 2d, questo valore si riferisce sempre ad un sistema di riferimento con assi che hanno origine nel centro dell'immagine

OUTPUT: point_3d *line, puntatore al vettore di elementi point_3d dove viene restituita la scansione

IplImage ***draw_2d**(point_3d *line)

Questa funzione visualizza graficamente il profilo 2d dei punti di una scansione. L'immagine generata è di tipo IPL_DEPTH_3U con dimensioni WxW pixel. Il rapporto fra le distanze dell'immagine è conservato rispetto al valore reale delle coordinate dei punti, ma è riscalato di un fattore 10.

INPUT: point_3d *line, puntatore al vettore di elementi point_3d dove della scansione

OUTPUT: point_3d *line, puntatore al vettore di elementi point_3d dove viene restituita la scansione il vettore contiene i 640 valori delle coordinate dei punti reali nello spazio espresse in millimetri, gli elementi del vettore si riferiscono ai punti della depth image con ascissa da 0 a W.

L'immagine restituita viene visualizzata posizionando l'origine di riferimento delle coordiante dei punti in alto e in centro alla finestra visualizzata. Questo ovviamente inverte l'ordine dei punti che nella realtà si trovano con l'ascissa invertita rispetto a quella dei punti nello spazio.

Si noti che questa funzione ha solo lo scopo di visualizzare graficamente la sezione di spazio scannerizzata al fine di dimostrare il corretto funzionamento del sistema.

3.4.3. test.c

Il file test.c contiene un esempio di applicazione delle funzioni precedentemente riportate. Il programma effettua una scansione continua della depth image secondo un piano con ordinata pari a zero, ovvero che taglia lo spazio a metà nel campo visivo del sensore. Inoltre include le funzioni necessarie per l'IPC fra altri processi concorrenti, che devono consumare i dati delle scansioni mediante il meccanismo della shared memory.

char ***sm_init**(int sm_id)

Inizializza il segmento di memoria condivisa, chiedendo al sistema operativo di allocare un segmento si grandezza SM_SIZE e con i permessi SM_PERM.

INPUT: int sm_id, valore intero che specifica lid del segmento di memoria condiviso, il medesimo valore deve essere utilizzato da un altro processo per accedere al medesimo segmento di memoria. OUTPUT: Il valore restituito è l'indirizzo di meoria del segmento fornito dalla syscall shmat().

Nel caso non risulti possibile allocare spazio per il segmento di memoria, la funzione termina restituendo un puntatore NULL, mentre nel caso non risulti possibile effettuare l'attach del segmento nello spazio di memoria del processo, la funzione termina restituendo l'output fornito dalla syscall shmat().

void **sm_copy**(char *sm_p, point_3d *line)

Copia nel segmento di memoria condivisa, SM_SIZE byte dalla zona di meoria referenziata dal puntatore line.

INPUT:

- char *sm_p: puntatore di tipo char al segmento di memoria condiviso
- point_3d *line: puntatore di tipo point_3d al vettore contenete la scansione

int **sm_free**(int sm_id, char *sm_p)

Funzione che si occupa di fare il *detach* e di chiedere la rimozione del segmento di memoria condivisa, con la conseguente perdita dei dati contenuti e della possibilità di *attach* da parte di altri processi concorrenti. Per ulteriori informazioni si rinvia l'utente alla documentazione delle IPC in ambiente UNIX.

INPUT:

- int sm_id: id del segmento di memoria condiviso da rimuovere
- char *sm_p: puntatore al segmento di memoria condiviso da rimuovere

OUTPUT: Nel caso non risulti possibile fare il detach di un segmento, la funzione restituisce -1, altrimenti la funzione termina con il valore zero.

Il resto del file contiene il codice che accede al sensore Kinect mediante il wrapper sincrono restituendo immagini dai sensori di tipo IplImage. La depth image viene filtrata con la funzione remove_noise() e successivamente viene fatta una scansione 2d sul piano con ordinata zero, infine i valori sono visualizzati graficamente tramite draw_2d().

4. Modalità operative

4.1. Componenti necessari

Gli strumenti necessari all'attuazione della soluzione adottata sono:

- Libreria Libfreenect
- Libreria ROS
- Libreria OpenCV

4.1.1. Libfreenect

Libfreenect è una libreria cross-platform open-source sviluppata dal progetto *OpenKinect*, che fornisce le interfacce necessarie ad attivare, inizializzare e comunicare dati con l'hardware del dispositivo Kinect. La libreria supporta l'accesso allo stream video della IR camera, dell'RGB camera, ai valori di disparità, impostare l'angolo del motore e lo stato dei LED, ottenuta come un reverse-engeegnering del protocollo di comunicazione e del firmware del sensore Kinect.

Mette inoltre a disposizione una serie di wrapper per fornire funzioni semplificate verso i principali linguaggi di programmazione.

4.1.2. ROS

Il sistema ROS (Robot Operating System) è un meta-operating system specifico al supporto multipiattaforma di robot. Si compone principalmente di un sistema di compilazione automatico per installare specifici pacchetti software sviluppati appositamente denominati *stack*.

Per quanto riguarda questo progetto si è sceglitelo di installare il pacchetto contenete le funzioni necessarie per la calibrazione del sensore Kinect, procedendo prima con la calibrazioni intrinseca e poi estrinseca delle due camere.

L'installazione di ROS è un processo complesso e non privo di errori, anche a causa della continua evoluzione del codice e del processo di sviluppo. La versione installata di ROS scelta per il processo di calibrazione è stata *fuerte* per GNU/Linux Ubuntu 12.04, in quanto è stata l'unica che si è riusciti ad installare correttamente. In particolare non si è stati riusciti ad installare ROS sul sistema MacOS X e a

compilarlo su un sistema GNU/Linux nella versione *groovy*. Si assume di seguito che la versione *ros-fuerte-ros-comm* sia installata nel sistema. Si noti che con la versione corrente *groovy* di ROS non è stato possibile effettuare la calibrazione a causa di bug nelle funzioni di calibrazione che bloccano l'applicazione non appena viene mostrata la checkerboard al Kinect.

4.1.3. OpenCV

OpenCV è una libreria open source destinata alla Visione Artificiale. Questa libreria, scritta in C e C++, sviluppata inizialmente da Intel, è disponibile su diverse piattaforme (Windows, Unix, MacOs e ultimamente anche per Android). L'ultima versione rilasciata ad oggi è la 2.4.5 , il sito dove poter scaricare le diverse SDK, con relativa documentazione, è il seguente :

http://opencv.willowgarage.com/wiki/ .

Le OpenCV sono suddivise in librerie :

- Cxcore: Strutture dati (array, matrici, elementi geometrici, etc..)con le relative operazioni (matematiche, logiche e algebriche);
- Cv: Manipolazioni delle immagini, analisi strutturale, inseguimento di oggetti (Object Tracking), riconoscimento (Pattern Recognition), Calibrazione delle videocamere e ricostruzioni di immagini in 3 dimensioni;
- Highgui: Interfacce utente grafiche (GUI) per la gestione delle immagini e dei video.
- Machine learning: Metodi di clustering e analisi dei dati;
- Cvaux:Metodi per la visione stereo e il tracking 3D;
- Cvcam: Interfacce per le videocamere.

Le funzioni messe a disposizione da OpenCV sono circa 500: elaborazione immagini, tracking e object detection, calibrazione dei dispositivi, estrazione delle feature da un'immagine, riconoscimento di volti e così via.

4.2. Modalità di installazione

4.2.1. Istallazione libreria Libfreenect

Si è scelto di installare le librerie libfreenect procedendo alla loro compilazione manuale, come riportato dalla sezione *Getting Started* nella wiki di OpenKinect.

Di seguito sono elencati i principali file inclusi nella libreria:

```
$ ls ./libfreenect/
APACHE20        CMakeFiles     CONTRIB   fakenect  include      README.asciidoc
build           CMakeLists.txt doc       GPL2      mkcontrib.sh src
CMakeCache.txt  cmake_modules  examples  HACKING   platform     wrappers
```

Di seguito vengono illustrati i principali passi per la compilazione e installazione della libreria libfreenect sui sistemi MacOS X e GNU/Linux Debian e Slackware.

4.2.1.1 Installazione su GNU/Linux

Per procedere con la compilazione sono necessarie le seguenti dipendenze software che devono essere già installate correttamente nel sistema:

- cmake
- libusb-1.0-0
- libusb-1.0-0-dev
- pkg-config
- libglut3
- libglut3-dev

Queste librerie sono richieste per la compilazione di libfreenect, è necessario controllare che siano presenti nel sistema altrimenti bisogna procedere alla loro installazione mediante sistema di gestione dei pacchetti, se presente, oppure mediante compilazione da sorgenti.

Durante l'installazione delle dipendenze su Debian è possibile che il pacchetto libglut3 non sia presente nel repository, questo pacchetto è stato rinominato in freeglut3.

$ sudo apt-get install cmake libglut3-dev pkg-config build-essential libxmu-dev libxi-dev libusb-1.0-0-dev

4.2.1.2 Installazione su Mac OSX

Su sistemi Apple le librerie richieste alla compilazione di libfreenect non sono presenti nel sistema standard, pertanto è necessario procedere alla loro installazione attraverso un sistema di porting come ad esempio MacPorts, inoltre deve essere presente il compilatore C e delle librerie standard libc, ad esempio installando Xcode.

I pacchetti che devono essere installati necessariamente dal port sono:

- git
- cmake

Dovrebbe essere possibile installare libusb-devel direttamente dal port, ma attualmente non risulta presente, pertanto si è dovuto procedere alla compilazione manuale:

$ git clone git://git.libusb.org/libusb.git

$ cd libusb

$ ./autogen.sh

Inoltre essendo libusb una libreria portata per i sistemi MacOS è necessario applicare la seguente patch:

$ patch -p1 < ../libfreenect/platform/osx/libusb-osx-kinect.diff

Si procede poi alla configurazione e alla compilazione e alla installazione:

$ ./configure LDFLAGS='-framework IOKit -framework CoreFoundation'

$ make

```
$ sudo make install
```

4.2.2. Compilazione della libreria Libfreenect

La libreria viene fornita mediante la clonazione del trunk di libfreenect:

```
$ git clone https://github.com/OpenKinect/libfreenect.git
$ mkdir build
$ cd build
```

Per procedere con la compilazione è necessario prima configurare correttamente il file CMakeLists.txt, dove sono riportate le direttive per la generazione del Makefile. Si procede quindi alla configurazione della compilazione:

```
$ ccmake ..
```

Si deve specificare correttamente dove sono collocate le librerie dinamiche e gli header file di libusb, inoltre è possibile anche definire quali parti delle libfreenect debbano essere incluse nel processo di compilazione.

```
BUILD_AS3_SERVER              OFF
BUILD_AUDIO                   OFF
BUILD_CPP                     ON
BUILD_CV                      OFF
BUILD_C_SYNC                  ON
BUILD_EXAMPLES                ON
BUILD_FAKENECT                ON
BUILD_PYTHON                  OFF
BUILD_REDIST_PACKAGE          OFF
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX          /usr/local
LIBUSB_1_INCLUDE_DIR          /usr/include/libusb-1.0
LIBUSB_1_LIBRARY              /usr/lib/i386-linux-gnu/libusb-1.0.so
LIB_SUFFIX
```

Una volta configurato, si procede con la generazione del Makefile e alla compilazione:

```
$ cmake ..
$ make
```

Successivamente si procede all'installazione delle librerie:

```
$ make install
```

Il processo di installazione provvede alla copia dei file delle librerie nelle directory destinazione del sistema operativo. Di particolare importanza è la locazione finale degli header file:

```
$ ls /usr/local/include/libfreenect
libfreenect.h  libfreenect-registration.h  libfreenect_sync.h
```

e degli object file:

```
$ ls /usr/local/lib/libfreenect*
/usr/local/lib/libfreenect.a          /usr/local/lib/libfreenect_sync.a
/usr/local/lib/libfreenect.so         /usr/local/lib/libfreenect_sync.so
/usr/local/lib/libfreenect.so.0.1     /usr/local/lib/libfreenect_sync.so.0.1
/usr/local/lib/libfreenect.so.0.1.2   /usr/local/lib/libfreenect_sync.so.0.1.2
```

Queste locazioni possono essere cambiate durante la configurazione mediante apposite opzioni da passare a cmake.

4.2.3. Errori nel caricamento delle librerie

Al fine di evitare errori relativi al non caricamento delle librerie installate nel sistema operativo da parte di applicazioni che le utilizzano, di seguito viene riportata una breve descrizione di come includere le locazioni delle librerie nella variabile di ambiente del caricatore dei programmi rilocabili.

Nei sistemi GNU/Linux per fare l'update delle librerie è sufficiente invocare il comando:

```
$ ldconfig
```

anche se, quando le librerie non vengono posizionate in locazioni standard che sono già incluse nella variabile di ambiente, queste non vengono comunque trovate. Risulta pertanto indispensabile aggiungerle in coda:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/some/library/path/
```

Inoltre è vantaggioso decidere di automatizzare questo processo aggiungendo il comando sopra riportato nel file di profilo della shell di sistema utilizzata, oppure aggiunger e il comando nel file */etc/profile*.

Nei sistemi MacOS la cosa è del tutto equivalente, ad eccezione del comando per aggiornare il caricatore:

```
$ export DYLD_LIBRARY_PATH=$ DYLD_LIBRARY_PATH: /usr/local/lib/libfreenect*
```

4.2.4. Udev

I sistemi GNU/Linux utilizzano un demone chiamato udev, che rileva i dispositivi hardware che vendono collegati al computer ed esegue determinate azioni.

Quando viene collegato il Kinect, devono essere creati dei rispettivi device nel sottosistema usb del kernel Linux. Il compito di udev in questo caso è solamente quello di impostare i permessi appropriati ai device file in modo che aia possibile utilizzare il Kinect anche in user space.

Per fare questo è necessario aggiungere un file che specifica il comportamento da seguire, nella directory delle regole di udev:

$ cat /etc/udev/rules.d/66-kinect.rules

ATTR{product}=="Xbox NUI Motor"

SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02c2", MODE="0666", GROUP="plugdev"

ATTR{product}=="Xbox NUI Audio"

SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad", MODE="0666", GROUP="plugdev"

ATTR{product}=="Xbox NUI Camera"

SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae", MODE="0666", GROUP="plugdev"

4.2.5. Installazione libreria OpenCV

Si consiglia di compilare e installare da sorgente la versione stabile più recente seguendo le indicazioni presenti a questo indirizzo http://opencv.willowgarage.com/wiki/InstallGuide.

4.2.6. Wrapper OpenCV e c_sync

La libreria libfreenect fornisce delle funzioni C che possono essere utilizzate direttamente dalle applicazioni, in maniera semplice senza doversi occupare di tutta la perate di inizializzazione e configurazione preliminare del dispositivo.

Il wrapper opencv fornisce due funzioni per l'accesso ai dati forniti dal sensore, che restituiscono l'output in forma di strutture IplImage della libreria OpenCV.

Le funzioni C sono contenute nel file *libfreenect_cv.c*:

IplImage *freenect_sync_get_depth_cv(int index)

IplImage *freenect_sync_get_rgb_cv(int index)

Queste due funzioni restituiscono un puntatore di tipo Iplimage alla matrice contenente i dati del sensore, rispettivamente della depth camera e della rgb camera. Le immagini restituite sono definite come static e create solamente durante la prima esecuzione della funzione, per tutte le altre esecuzioni, le immagini vengono riutilizzate, essendo sovrascritte con i nuovi dati provenienti dal sensore.

Il formato dell'immagine di profondità restituita dalla prima funzione, consiste in una matrice di 640x480 ad un solo canale e di tipo IPL_DEPTH_16U, ovvero 16bit unsigned.

Il formato dell'immagine RGB restituita dalla seconda funzione è di 640x480 a 3 canali e di tipo IPL_DEPTH_8U, ovvero 8bit di profondità.

Il wrapper c_sync fornisce una interfaccia sincrona dei dati provenienti dal sensore mediante l'uso di chiamate bloccanti. Il wrapper è costituito da un thread che fornisce le funzioni di call-back alla libreria libfreenect, e di un buffer dove vengono salvati temporaneamente i dati degli stream. Il wrapper

garantisce l'ordine monotonico dei dati nel buffer mediante il meccanismo dei timestamp, ma non garantisce la completa uniformità dei campioni, ovvero ogni tanto possono capitare dei salti fra un campione ed il successivo.

Le principali funzioni contenute nel wrapper c_sync:

```
int freenect_sync_get_depth(    void **depth,
                                uint32_t *timestamp,
                                int index, freenect_depth_format fmt)

int freenect_sync_get_video(    void **video,
                                uint32_t *timestamp,
                                int index,
                                freenect_video_format fmt)
```

più altre funzioni per gestire il tilt del sensore.

Queste due funzioni permettono di specificare in quale modalità si vuole fare funzionare il sensore, nel wrapper originale di libfreenect la funzione freenect_sync_get_depth(), per la depth camera, fornisce il valore raw a 11bit della disparità letta direttamente dalla misurazione stereoscopia del Kinect, mentre freenect_sync_get_video() fornisce l'immagine della rgb camera.

4.2.6.1 Compilazione del wrapper OpenCV

La compilazione del wrapper avviene semplicemente attraverso l'utilizzo di cmake:

```
$ cd ./libfreenect/wrappers/opencv/
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Questo compila il programma *cvdemo.c* di esempio contenuto nel wrapper. La compilazione dovrebbe avvenire senza particolari errori, ad eccezione dei sistemi MacOS X, in cui è previsto di modificare il file CMakeLists.txt come segue:

```
##############################################################################
##
# OpenCV Interface
##############################################################################
##
find_package(OpenCV REQUIRED)
add_library (freenect_cv SHARED libfreenect_cv.c)
set_target_properties (freenect_cv PROPERTIES
```

```
  VERSION ${PROJECT_VER}
  SOVERSION ${PROJECT_APIVER})

include_directories (../c_sync)

target_link_libraries (freenect_cv freenect_sync ${OpenCV_LIBS})

install (TARGETS freenect_cv
  DESTINATION "/usr/local/lib/libfreenect*")
install (FILES "libfreenect_cv.h"
  DESTINATION "/usr/local/include/libfreenct*")

add_executable(cvdemo cvdemo.c)
target_link_libraries(cvdemo freenect freenect_sync freenect_cv ${OpenCV_LIBS})
install (TARGETS cvdemo
  DESTINATION bin)
```

4.3. Calibrazione del sensore con ROS

Per procedere alla calibrazione del Kinect con ROS sono necessari i seguenti stack:

- openni_camera
- openni_launch
- camera_calibration
- camera_pose
- openni_kinect

Per una installazione più dettagliata degli stack, si rimanda al tutorial per la camera calibration della documentazione di ROS.

La procedura di calibrazione si compone di due fasi, una calibrazione intrinseca di entrambe le camere RGB e IR ed una calibrazione estrinseca delle due camere.

Come primo passo è necessario avviare il driver OpenNI:

```
$ roslaunch openni_launch openni.launch
```

Nei messaggi di output visualizzati si deve essere certi che il nodo rieca a trovare il dispositivo connesso:

```
process[camera_base_link3-22]: started with pid [8850]
```

[INFO] [1370502009.937321367]: Number devices connected: 1

[INFO] [1370502009.938570060]: 1. device on bus 001:08 is a Xbox NUI Camera (2ae) from Microsoft (45e) with serial id '0000000000000000'

[INFO] [1370502009.941589438]: Searching for device with index = 1

[INFO] [1370502010.017930498]: Opened 'Xbox NUI Camera' on bus 1:8 with serial number '0000000000000000'

[INFO] [1370502010.092052914]: rgb_frame_id = '/camera_rgb_optical_frame'

[INFO] [1370502010.092375511]: depth_frame_id = '/camera_depth_optical_frame'

Spesso si è costretti a terminare questo servizio mediante ctrl-c, si consiglia sempre di verificare che il server contenga i driver per la kinect venga anch'esso terminato:

```
$ ps -e | grep XnSensorServer
 8190 ?        00:00:00 XnSensorServer
$ kill -9 8190
```

La mancata terminazione di questo servizio ha causato alcuni errori durante il processo successivo riavvio del servizio openni_launch. Durante l'esecuzione di questo nodo sono numerosi i messaggi di errore che vengono stampati a video, non tutti però sono fatali e nonostante la loro presenza si riesce lo stesso ad avviare il processo di calibrazione.

Successivamente si verifica che il nodo si sia avviato correttamente:

```
$ rostopic list
```

Per accertarsi che ROS abbia correttamente rilevato il sensore Kinect e pubblicato tutte le informazioni legate alle camere, in particolare di seguito sono riportate quelle relative alla depth camera:

/camera/depth/camera_info

/camera/depth/disparity

/camera/depth/image

/camera/depth/image_raw

/camera/depth/image_rect

/camera/depth/image_rect_raw

alcune relative alla IR camera:

/camera/ir/camera_info

/camera/ir/image_raw

/camera/ir/image_rect

ed alcune relative alla RGB camera:

/camera/rgb/camera_info

/camera/rgb/debayer/parameter_descriptions

/camera/rgb/debayer/parameter_updates

/camera/rgb/image_color

/camera/rgb/image_mono

/camera/rgb/image_raw

/camera/rgb/image_rect

/camera/rgb/image_rect_color

/camera/rgb/rectify_color/parameter_descriptions

Inoltre per procedere con la calibrazione intrinseca si deve disporre di una checkerboard stampata su un supporto rigido.

Come primo passo si è eseguita la calibrazione della RGB camera:

$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.4 image:=/camera/rgb/image_raw camera:=/camera/rgb

specificando mediante –size il numero di fronti bianco-nero interni alla checkerboard sia verticali (cols) che orizzontali (rows), esclusi i bordi, mentre con l'attributo –square si specifica il lato in metri del quadrato della scacchiera.

ROS deve rispondere trovando il servizio:

Waiting for service /camera/rgb/set_camera_info ...

OK

Ecco mostrata in figura la finestra che viene visualizzata se si è eseguito il comando correttamente:

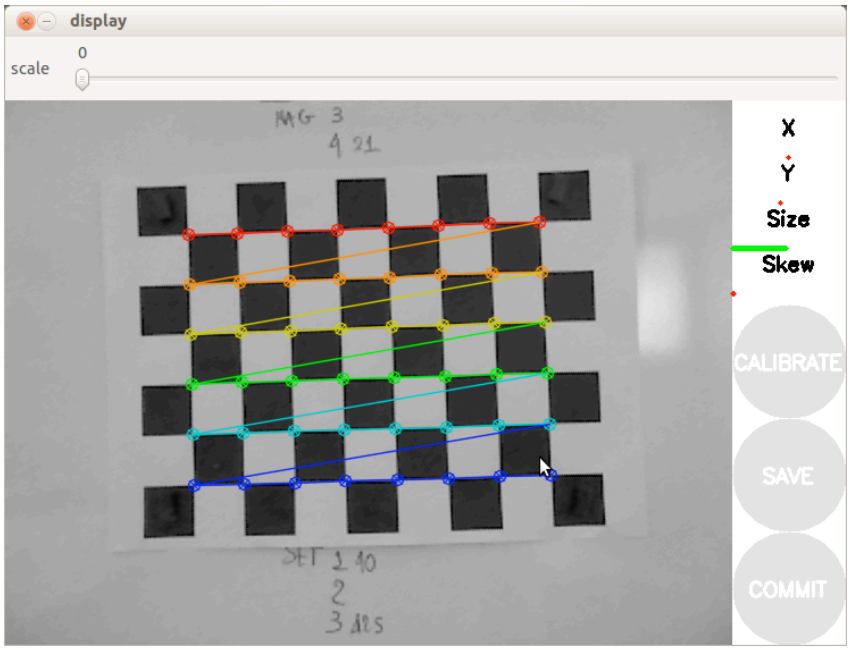

Fig. 13 - Calibrazione della RGB camera

La stessa cose deve essere ripetuta per l'IR camera, modificando la risorsa da utilizzare:

$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.4 image:=/camera/ir/image_raw camera:=/camera/ir

Come si nota dalla figura , per fare si che le funzioni di OpenCV per la calibrazione della IR camera riescano ad individuare la checkerboard, l'illuminatore laser deve essere coperto, altrimenti non si è in grado di effettuare il processo.

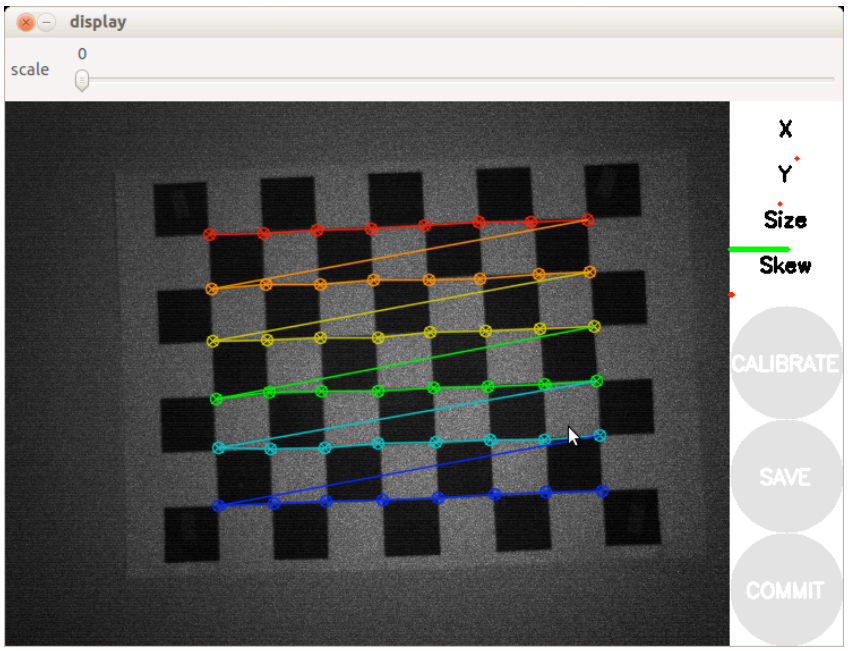

Fig. 14 - Calibrazione della IR camera

La calibrazione per la RGB camera e la IR camera avviene mostrando al Kinect la checkerboard, orientata in più posizioni, continuando fino a che le barre colorate riportate in fianco alla finestra non sono tutte verdi o fino a quando non risulta evidenziato il bottone per la calibrazione; i risultati sono mostrati a video e possono essere salvati in apposito file. Al termine bisogna fare il commit delle calibrazioni per pubblicare ai servizi di ROS i risultati ottenuti.

Con un procedimento simile dovrebbe essere possibile effettuare anche la calibrazione estrinseca fra la RGB camera e l'IR camera per determinare la matrice di rototraslazione per passare dalle coordinate di una camera a quelle dell'altra.

Questo processo non viene descritto poiché non si è riusciti a completarlo, anche seguendo la documentazione di ROS alla voce Calibrating the Kinect depth camera to the built-in RGB camera. Il firmware del Kinect non permette di ottener lo stream dalla RGB camera e dalla IR camera contemporaneamente, pertanto è necessario effettuare una registrazione dello stream RGB che deve essere in seguito proiettato in sincrono a quello IR. In questa seconda fase il processo non risulta in grado di resettare correttamente le funzioni OpenCV per la calibrazione, che non individuano la checkerboard, andando di conseguenza in time-out. Per risolvere questo problema sono stati vagliati diversi work-around, come l'utilizzo di una diversa versione delle librerie OpenCV, o l'utilizzo di un apposito stack denominato *contrast_augmenter*, per aumentare il contrasto della checkerboard nello stream IR, ma nessuno di questi tentativi è riuscito a risolvere il problema, si dunque deciso di procedere utilizzando solamente la calibrazione estrinseca.

4.4. Esempo di utilizzo delle applicazioni realizzate

Per utilizzare il programma di esempio è prima necessario procedere alla sua compilazione mediante cmake. Di seguito è riportato il CMakeLists.txt incluso nella directory dell'applicazione:

```
##############################################################################
##

# Kinect

##############################################################################
##



cmake_minimum_required(VERSION 2.8)
find_package(OpenCV REQUIRED)


add_library (freenect_cv SHARED libfreenect_cv.c)

add_library (scan SHARED scan.c)



set_target_properties (freenect_cv PROPERTIES

  VERSION ${PROJECT_VER}

  SOVERSION ${PROJECT_APIVER})
```

```
set_target_properties (scan PROPERTIES

  VERSION ${PROJECT_VER}

  SOVERSION ${PROJECT_APIVER})


include_directories (/usr/local/include/libfreenect/)

target_link_libraries (scan freenect_cv freenect_sync ${OpenCV_LIBS})


install (TARGETS scan DESTINATION bin)

install (TARGETS freenect_cv DESTINATION bin)

install (FILES "libfreenect_cv.h"  DESTINATION bin)

install (FILES "scan.h"  DESTINATION bin)


add_executable(test test.c)

target_link_libraries(test scan ${OpenCV_LIBS})

install (TARGETS test DESTINATION bin)


add_executable(sm sm.c)

target_link_libraries(sm scan ${OpenCV_LIBS})

install (TARGETS sm DESTINATION bin)
```

Per la compilazione dell'esempio è richiesta l'inclusione del path della directory contenete gli header file della libreria libfreenect, questo punto risulta essenziale per la corretta compilazione.

Il processo genera due file eseguibili:

- test

Il programma di esempio che utilizza direttamente la libreria scan.c per effettuare la scansione e visualizzarla graficamente a video.

- sm

Un semplice programma che illustra il meccanismo di accesso al segmento di memoria condiviso per la lettura dei dati forniti dalla scansione.

Di seguito è riportato un esempio di come viene visualizzata la scansione e della relativa immagine RGB:

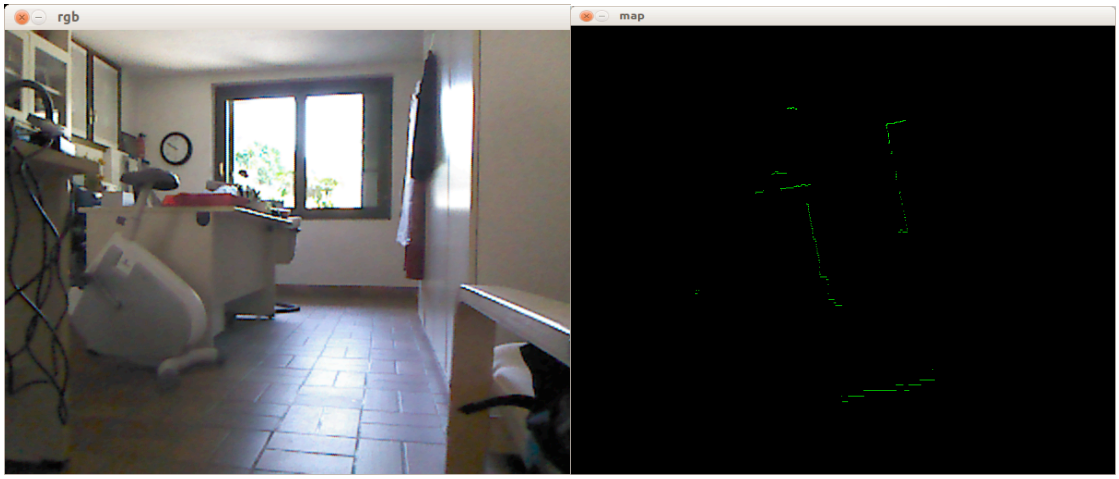

Fig. 15 - Esempio applicazione scanner 2D

5. Conclusioni e sviluppi futuri

In conclusione si è raggiunto lo scopo realizzare una libreria che permette di utilizzare il sensore Kinect per effettuare scansioni 2D dell'ambiente circostante e di poterla sfruttare da processi esterni attraverso una modalità di scambio dati inter processo con l'utilizzo della memoria condivisa.

Attraverso l'uso delle libfreenect si è riusciti ad ottenere una depth image già opportunamente convertita in millimetri ed anche correttamente allineata all'immagine RGB. Non essendo stato possibile effettuare la calibrazione estrinseca del sensore, non si è neanche di conseguenza, potuto ottenere una validazione dell'allineamento fra le due immagini. Si noti che il problema relativo all'allineamento è di secondaria importanza relativamente allo scopo di questo elaborato, poiché la scansione 2D non lo richiede necessariamente.

Il processo di calibrazione intrinseca della camera IR del sensore è stato fondamentale per riuscire ad ottenere un primo calcolo delle coordinatre reali dei punti nello spazio, anche se non si è effettuato una successiva analisi per determinare l'effettiva correttezza.

Un sensibile miglioramento si è ottenuto invece per quanto riguarda il processo di filtraggio della depth image in quanto si è riusciti ad eliminare in buona approssimazione il rumore introdotto dal sensore.

Possibili sviluppi futuri possono prendere in considerazione:

- una più accurata validazione del calcolo delle coordinate reali dei punti
- il problema della calibrazione estrinseca del sensore per riuscire ad ottenere un allineamento fra le due camere
- la perdita di uniformità dei profili di oggetti lontani dal sensore durante la scansione

Appendice A

Shared Memory

Shared Memory is an efficeint means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access.

In the Solaris 2.x operating system, the most efficient way to implement shared memory applications is to rely on the mmap() function and on the system's native virtual memory facility. Solaris 2.x also supports System V shared memory, which is another way to let multiple processes attach a segment of physical memory to their virtual address spaces. When write access is allowed for more than one process, an outside protocol or mechanism such as a semaphore can be used to prevent inconsistencies and collisions.

A process creates a shared memory segment using shmget()|. The original owner of a shared memory segment can assign ownership to another user with shmctl(). It can also revoke this assignment. Other processes with proper permission can perform various control functions on the shared memory segment using shmctl(). Once created, a shared segment can be attached to a process address space using shmat(). It can be detached using shmdt() (see shmop()). The attaching process must have the appropriate permissions for shmat(). Once attached, the process can read or write to the segment, as allowed by the permission requested in the attach operation. A shared segment can be attached multiple times by the same process. A shared memory segment is described by a control structure with a unique ID that points to an area of physical memory. The identifier of the segment is called the shmid. The structure definition for the shared memory segment control structures and prototypews can be found in <sys/shm.h>.

Bibliografia

[1] Myron Z. Brown, Darius Burschka e Gregory D. Hager, "Advances in Computational Stereo IEEE Transactions on Pattern Analysis and Machine Intelligence", vol. 25, num. 8, pag. 993-1008, Agosto 2003

[2] J. Banks e P. Corke, "Quantitative Evaluation of Matching Methods and Validity Measures for Stereo Vision", *Int'l J. Robotics Research*, vol. 20, num. 7, 2001

[3] Kudo ; Fitzgibbons; Microsoft Corp. Redmond WA. "Kinect for XBox 360"

[4] M.R. Andersen, T. Jensen, P. Lisouski, A.K. Mortensen, M.K. Hansen, T. Gregersen and P. Ahrendt: "Kinect Depth Sensor Evaluation for Computer Vision Applications", *Aarhus University, Department of Engineering* 2012

Web resources:

[1] http://en.wikipedia.org/wiki/Kinect

[2] http://www.microsoft.com

[3] http://www.openni.org

[4] http://www.openkinect.org

[5] http://www.ros.org/wiki/kinect

[6] http://www.ros.org/wiki/kinect_calibration/technical

[7] "How kinect depth sensor works. stereo triangulation?" http://mirror2image.wordpress.com/2010/11/30/how-kinect-works-stereo-triangulation

[8] "Prime sense kinect's hardware" http://www.primesense.com, 2010

[9] http://www.cs.cf.ac.uk/Dave/C/node27.html

Indice