

<b>INTRODUZIONE</b>	<b>2</b>
<b>KINECT</b>	<b>3</b>
Specifiche tecniche .....	3
Progetto OpenKinect.....	4
Calibrazione .....	5
Creazione della point cloud.....	7
Libreria libMyKinect.....	9
<b>GRAFFITI DETECTION</b>	<b>11</b>
Background Initialization .....	11
Cambiamenti di colore e di posizione .....	11
Analisi dei graffiti.....	13
Risultati sperimentali.....	13
Problemi riscontrati .....	14
<b>HAND GESTURE</b>	<b>15</b>
Estrazione del braccio.....	15
Determinazione del colore della mano .....	15
Individuazione della mano .....	18
Analisi della posizione e dell'orientamento .....	20
Risultati sperimentali.....	21
Problemi riscontrati .....	22
<b>CONCLUSIONI</b>	<b>23</b>
<b>APPENDICE</b>	<b>24</b>
Compilazione del software .....	24
OpenNI.....	24

# INTRODUZIONE

Microsoft Kinect è un accessorio per Xbox 360 che permette di interagire con la console senza l'uso di ulteriori dispositivi, come invece accade per le concorrenti Wii e Ps3.



Ufficialmente l'utilizzo di questa periferica è limitato alla sola console Xbox 360, ma un gruppo di programmatore è riuscito a mettere a punto un driver funzionante anche per PC, dando vita a numerosi progetti non ufficiali portati avanti da una folta comunità.

In questo progetto si vogliono realizzare due applicazioni che utilizzando il Kinect permettano di effettuare il riconoscimento di graffiti e il controllo del mouse di un PC attraverso gesture della mano.

Nei capitoli successivi saranno quindi descritti l'hardware e il funzionamento del Kinect. Si mostrerà poi la libreria freenect utilizzata nel progetto e gli strumenti utilizzati per effettuare la calibrazione del dispositivo. Seguirà infine una descrizione dettagliata dei due programmi realizzati e dei risultati sperimentali ottenuti.

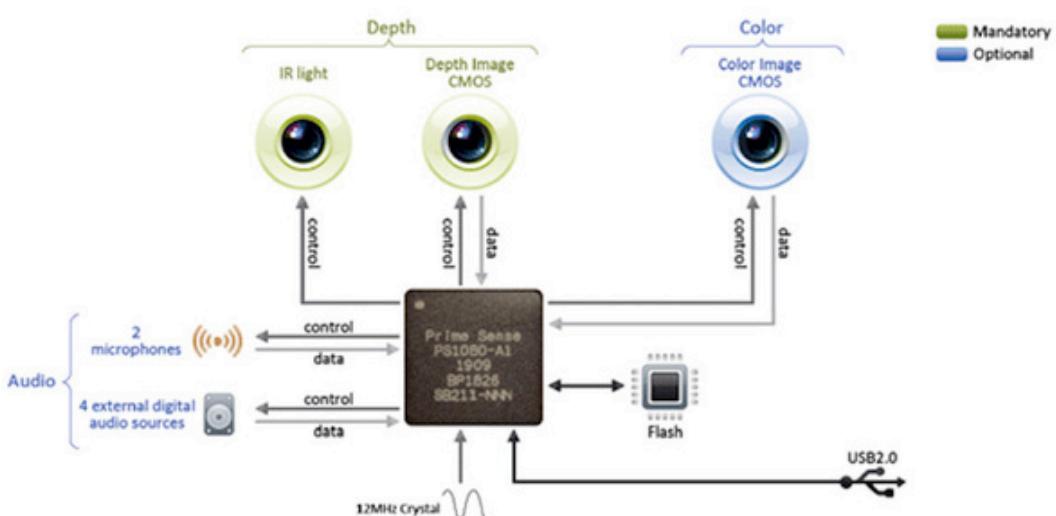
# KINECT

## Specifiche tecniche

Il Kinect è dotato di una telecamera RGB e di un sensore di profondità composto da un proiettore a infrarossi e da una telecamera sensibile alla stessa banda. Le due telecamere hanno una risoluzione di  $640 \times 480$  pixel e forniscono immagini a 30fps. L'immagine seguente mostra l'hardware interno del dispositivo.



Oltre ai sensori è visibile la scheda elettronica contenente il sistema di elaborazione interno: l'innovativa tecnologia sviluppata da PrimeSense non richiede infatti l'applicazione a valle di algoritmi di matching stereo - come nel caso di telecamere stereoscopiche - ma effettua direttamente in hardware una parte delle operazioni, rendendo direttamente disponibili i dati sulla distanza degli oggetti.



Il Kinect dispone anche di quattro microfoni che sono orientati in direzioni diverse e vengono utilizzati dal sistema per la calibrazione dell'ambiente in cui ci si trova, tenendo conto del modo in cui il suono rimbalza sulle pareti e sui mobili. La barra del Kinect è motorizzata lungo l'asse verticale e segue i movimenti dei giocatori, orientandosi nella posizione migliore per il riconoscimento dei movimenti. Di fatto, 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. Per un corretto funzionamento del dispositivo non è sufficiente l'alimentazione fornita da una porta USB standard, per cui è necessario l'impiego di un alimentatore dedicato.

## Progetto OpenKinect



I driver utilizzati nei due programmi non sono stati rilasciati ufficialmente da PrimeSense, ma sono stati sviluppati dalla comunità tramite tecniche di reverse engineering

nell'ambito del progetto OpenKinect. Per questo motivo i driver non permettono il pieno controllo del dispositivo. Tuttavia tramite la libreria *freenect* è possibile acquisire correttamente le immagini provenienti da entrambi i sensori video (RGB e Depth).

In particolare il driver è basato sulla libreria libusb tramite la quale gestisce lo streaming di dati proveniente dal Kinect. È possibile quindi effettuare la registrazione di apposite funzioni di callback che saranno automaticamente invocate quando il dispositivo ha acquisito una nuova immagine RGB o Depth. L'approccio basato su callback tuttavia introduce notevoli complicazioni, poiché richiede l'accesso ad un'area dati condivisa che deve essere quindi opportunamente gestito tramite semafori.

Per semplificare la vita agli sviluppatori vengono quindi forniti diversi *wrapper* che, oltre a permettere l'utilizzo dei driver con linguaggi di programmazione diversi dal C, consentono il recupero delle immagine tramite l'invocazione di chiamate sincrone a funzioni.

Nel progetto si è quindi utilizzato il wrapper chiamato *c\_sync*, il quale mette a disposizione le chiamate alle seguenti funzioni:

```
int freenect_sync_get_video(void **video, uint32_t *timestamp, int index,
freenect_video_format fmt);
/* Synchronous video function, starts the runloop if it isn't running

    The returned buffer is valid until this function is called again, after which the
buffer must not
    be used again. Make a copy if the data is required.

Args:
    video: Populated with a pointer to a video buffer with a size of the requested
type
    timestamp: Populated with the associated timestamp
    index: Device index (0 is the first)
    fmt: Valid format

Returns:
    Nonzero on error.
*/
```

```
int freenect_sync_get_depth(void **depth, uint32_t *timestamp, int index,
freenect_depth_format fmt);
/* Synchronous depth function, starts the runloop if it isn't running
```

```

The returned buffer is valid until this function is called again, after which the
buffer must not
be used again. Make a copy if the data is required.

Args:
    depth: Populated with a pointer to a depth buffer with a size of the requested
type
    timestamp: Populated with the associated timestamp
    index: Device index (0 is the first)
    fmt: Valid format

Returns:
    Nonzero on error.
*/

```

Entrambe le funzioni allocano una quantità di memoria sufficiente a contenere le informazioni richieste. In particolare la camera RGB fornisce per ognuno dei punti 3 byte corrispondenti ai valori RGB, mentre la camera depth fornisce valori di 11 bit proporzionali alla distanza misurata. Al termine dell'utilizzo dei dati è opportuno liberare la memoria allocata attraverso una free, per evitare di saturare rapidamente la RAM disponibile nel sistema. Nel prossimo paragrafo si mostrerà come utilizzare questi dati per stimare la posizione dei punti nello spazio.

I driver sono open-source e possono essere utilizzati con Windows, Linux e Mac OSX. Maggiori informazioni sul progetto sono disponibili sul sito internet ufficiale: <http://openkinect.org/>

Il codice sorgente è accessibile dal repository: <https://github.com/OpenKinect/libfreenect>

## Calibrazione

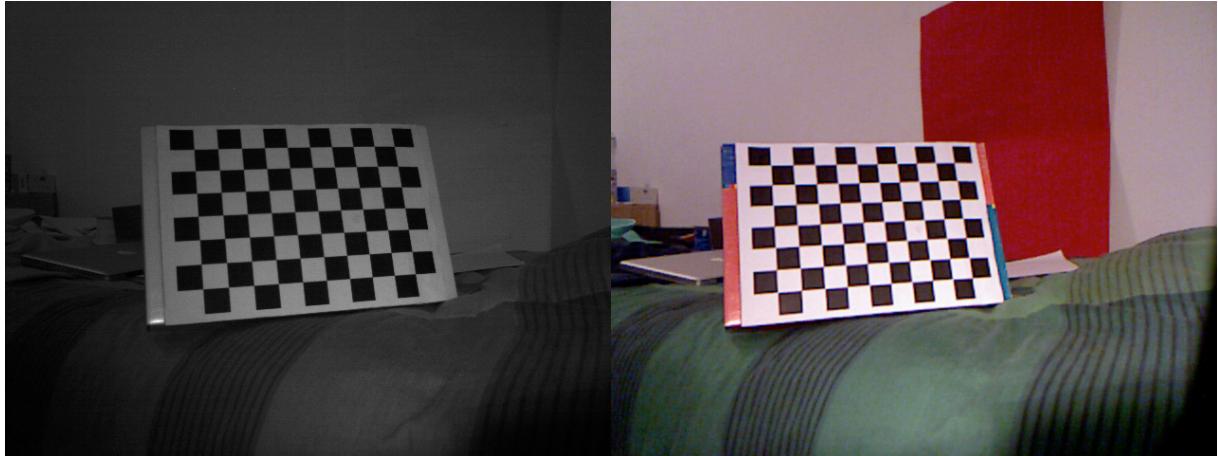
Prima di poter iniziare ad utilizzare il dispositivo all'interno di un progetto è necessario procedere alla calibrazione delle due telecamere presenti, al fine di determinare la rototraslazione tra i due sensori e poter poi mappare correttamente i punti dell'immagine a colori su quella della profondità. In questo modo è possibile ottenere una point cloud colorata sulla quale applicare successivamente algoritmi di computer vision.

Il software utilizzato per la calibrazione sfrutta solamente la libreria OpenCV ed è reperibile a questo indirizzo: <http://www.informatik.uni-freiburg.de/~engelhar/calibration.html>

Tale software è in grado di analizzare una serie di coppie di immagini (RGB e Depth) precedentemente acquisite. Il driver freenect non permette lo streaming simultaneo delle due immagini perciò è fondamentale che la checkerboard sia il più possibile ferma.

Nell'ambito di questo progetto il set di immagini è stato acquisito con il software sviluppato da Nicolas Burrus. Maggiori informazioni disponibili alla seguente pagina: <http://nicolas.burrus.name/index.php/Research/KinectRgbDemo>

Le immagini seguenti sono state catturate coprendo l'emettitore infrarossi e illuminando l'oggetto con una lampada alogena, come suggerito dal sito del ROS alla seguente pagina: <http://www.ros.org/wiki/kinect/Tutorials/Calibrating%20the%20Kinect>. In questo modo si ottengono immagini idonee alla calibrazione.



Lo strumento di calibrazione richiede un set di immagini e provvede ad estrarre i parametri in modo completamente automatico. Esistono anche altri strumenti per la calibrazione, come quello proposto dallo stesso Nicolas Burrus, che richiedono all'utente di selezionare con il mouse alcuni punti significativi dell'immagine, risultando ovviamente meno precisi. L'output viene fornito attraverso i seguenti file xml, posti nella cartella /data:

```
=====
Filename:      data/Distortion_IR.xml
=====

<?xml version="1.0"?>
<opencv_storage>
<Distortion_IR type_id="opencv-matrix">
<rows>5</rows>
<cols>1</cols>
<dt>f</dt>
<data>
    -0.27463219  0.83871746  4.19331342e-03  -3.12518585e-03  0.</data></
</Distortion_IR>
</opencv_storage>

=====

Filename:  data/Distortion_RGB.xml
=====

<?xml version="1.0"?>
<opencv_storage>
<Distortion_RGB type_id="opencv-matrix">
<rows>5</rows>
<cols>1</cols>
<dt>f</dt>
<data>
    0.09779982  -0.03481068  0.01980981  -1.80149463e-03  0.</data></
</Distortion_RGB>
</opencv_storage>

=====

Filename:  data/Intrinsics_IR.xml
=====

<?xml version="1.0"?>
<opencv_storage>
<Intrinsics_IR type_id="opencv-matrix">
<rows>3</rows>
<cols>3</cols>
<dt>f</dt>
<data>
```

```

570.55841064 0. 323.78082275 0. 570.97265625 268.92361450 0. 0. 1.</data></
Intrinsics_IR>
</opencv_storage>

=====
Filename: data/Intrinsics_RGB.xml
=====
<?xml version="1.0"?>
<opencv_storage>
<Intrinsics_RGB type_id="opencv-matrix">
<rows>3</rows>
<cols>3</cols>
<dt>f</dt>
<data>
509.48260498 0. 305.60083008 0. 511.50158691 275.12200928 0. 0. 1.</data></
Intrinsics_RGB>
</opencv_storage>

=====
Filename: data/kinect_R.xml
=====
<?xml version="1.0"?>
<opencv_storage>
<kinect_R type_id="opencv-matrix">
<rows>3</rows>
<cols>3</cols>
<dt>f</dt>
<data>
0.99999845 9.63805651e-04 -1.48875453e-03 -9.64035571e-04 0.99999952
-1.53721863e-04 1.48860563e-03 1.55156828e-04 0.99999887</data></kinect_R>
</opencv_storage>

=====
Filename: data/kinect_T.xml
=====
<?xml version="1.0"?>
<opencv_storage>
<kinect_T type_id="opencv-matrix">
<rows>3</rows>
<cols>1</cols>
<dt>f</dt>
<data>
-2.48245621 -0.01868717 0.08752445</data></kinect_T>
</opencv_storage>
```

Questi parametri sono stati ottenuti utilizzando una checkerboard in formato A4 e un set di 20 immagini. Il contenuto dei file rispetta il formato standard di OpenCV e può essere facilmente letto tramite la funzione cvLoad.

## Creazione della point cloud

Una volta ricavati i parametri delle due camere è possibile costruire una point cloud a colori. La prima operazione da fare è proiettare i punti della telecamera Depth nello spazio 3D.

Per ogni punto è infatti possibile stimare la distanza dalla camera utilizzando formule ricavate sperimentalmente. Di seguito sono riportate due diverse funzioni per il calcolo della distanza in metri a partire dal valore di 11 bit nell'immagine depth.

```

float raw_depth_to_meters(int raw_depth) {
    if (raw_depth < 2047) {
        return 1.0 / (raw_depth * -0.0030711016 + 3.3309495161);
    }
    return 0;
}

float raw_depth_to_meters(int raw_depth) {
    if (raw_depth < 2047) {
        const float k1 = 1.1863;
        const float k2 = 2842.5;
        const float k3 = 0.1236;
        return k3 * tanf(raw_depth/k2 + k1);
    }
    return 0;
}

```

All'interno del progetto è stato utilizzato il secondo metodo, poiché si è rivelato più preciso. Inoltre, per accelerare il calcolo delle distanze i valori sono stati inseriti in una Look Up Table che viene riempita una volta sola all'avvio del programma.

Si noti che non sempre il Kinect è in grado di determinare la distanza di un punto. In particolare se il valore di depth è 2048 il punto non è da considerarsi valido. Questa situazione si verifica ad esempio se gli oggetti sono posti ad una distanza troppo bassa (<40cm) dal sensore.

Una volta ottenuta la distanza è possibile individuare le coordinate spaziali del punto (d\_x, d\_y) come segue:

$$\begin{aligned}
 P3D.x &= (x_d - cx_d) * depth(x_d, y_d) / fx_d \\
 P3D.y &= (y_d - cy_d) * depth(x_d, y_d) / fy_d \\
 P3D.z &= depth(x_d, y_d)
 \end{aligned}$$

Dove fx\_d, fy\_d, cx\_d e cy\_d sono i parametri intrinseci della camera depth.

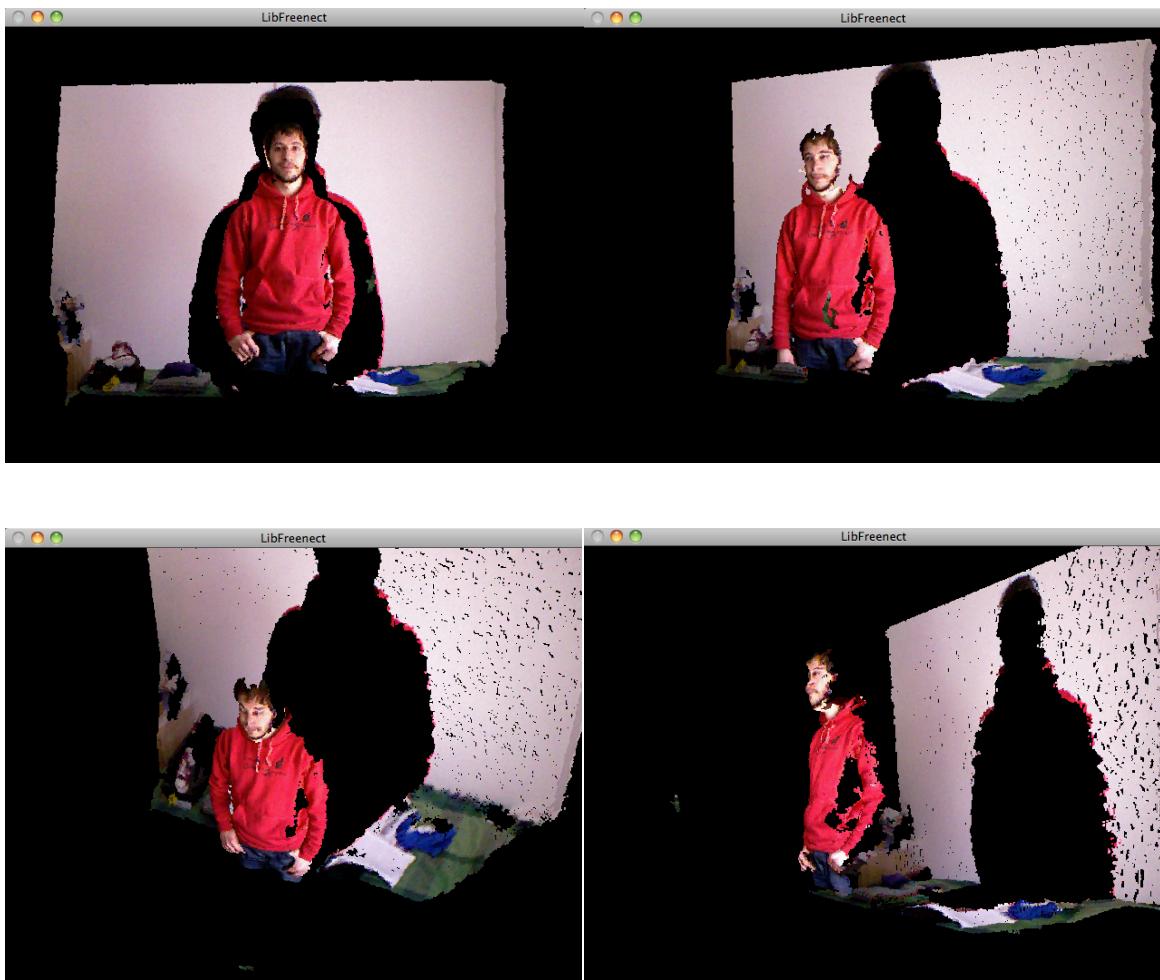
Possiamo quindi riproiettare il punto 3D sull'immagine a colori per estrarne il colore.

$$\begin{aligned}
 P3D' &= R * P3D + T \\
 P2D_rgb.x &= (P3D'.x * fx_rgb / P3D'.z) + cx_rgb \\
 P2D_rgb.y &= (P3D'.y * fx_rgb / P3D'.z) + cy_rgb
 \end{aligned}$$

R e T sono le matrici di rotazione e traslazione tra i due sensori mentre fx\_rgb, fx\_rgb, cx\_rgb e cy\_rgb sono i parametri intrinseci della camera RGB.

Le coordinate immagine P2D\_rgb.x e P2D\_rgb.y individuano il punto dell'immagine RGB corrispondente allo stesso punto della scena catturato dalla camera depth.

Con il metodo sopra descritto si può facilmente ottenere quindi una rappresentazione 3D e a colori della scena. Il programma d'esempio glpclview, sfruttando la libreria OpenGL, permette di visualizzare la point cloud, come mostrato nelle immagini seguenti, muovendo il punto di vista a piacimento.



## Libreria libMyKinect

La libreria libMyKinect è stata sviluppata nel corso del progetto per fornire un insieme di funzionalità di base per accedere ai dati forniti dal dispositivo in modo semplice. La libreria viene utilizzata da entrambi i programmi sviluppati e può rappresentare il punto di partenza anche per futuri lavori.

Le due principali strutture dati sono Point3d e PointCloud, definite come segue

```
typedef struct {
    int valid;
    float x;
    float y;
    float z;
    unsigned char blue;
    unsigned char green;
    unsigned char red;
    float Y;
    float U;
    float V;
} Point3d;

typedef Point3d PointCloud[480][640];
```

`Point3d` rappresenta un singolo punto della scena, per il quale si individuano le coordinate spaziali in metri e il colore espresso negli spazi RGB e YUV. Se il flag `valid` è uguale a 0, il punto non è valido e i restanti campi non sono significativi.

`PointCloud` invece è un insieme di `Point3d` e viene costruito tramite la funzione

```
int createPointCloud(PointCloud dest);
```

Questa funzione recupera i dati dal Kinect ed esegue tutte le operazioni descritte nel paragrafo precedente. I dati sulla calibrazione del dispositivo devono essere inseriti nella cartella `/data` e saranno caricati all'avvio del programma dalla funzione `loadParameters`.

La libreria contiene poi funzioni per ottenere una rappresentazione della point cloud su un'immagine 2D di tipo `IplImage`:

- `rgbImage`: crea un'immagine RGB a partire dalla point cloud. I punti non validi sono colorati di bianco.
- `depthImage`: crea un'immagine in scala di grigi nella quale la tonalità del grigio è proporzionale alla distanza misurata dal sensore `depth`.
- `binaryImage`: crea un'immagine binaria in cui i punti validi sono bianchi e i punti non validi sono neri.

Infine sono presenti altre funzioni ausiliarie che permettono ad esempio di misurare la distanza tra punti, calcolare il baricentro di una point cloud, ecc...

# GRAFFITI DETECTION

Il programma per la graffiti detection consente di individuare atti vandalici effettuati su una parete, come disegni o pitture. Utilizzando una semplice telecamera questo compito risulta estremamente difficile poiché non si riesce a distinguere un oggetto fermo davanti al muro da un disegno.

Grazie alle informazioni sulla profondità fornite dal Kinect è invece possibile risolvere facilmente questo tipo di problema: la costruzione di una point cloud infatti consente di individuare variazioni di colore e di spazio nella scena ripresa.

Nei paragrafi seguenti sarà descritto il funzionamento del programma.

## Background Initialization

Il sistema di change detection implementato è basato su *background subtraction* quindi è necessario in primo luogo acquisire un background di riferimento. L'operazione deve essere effettuata in assenza totale di individui nella scena.

La cattura del background viene avviata tramite la pressione del tasto x e può essere ripetuta in qualunque momento durante l'esecuzione del programma. Con la funzione `backgroundInit` si costruisce ad intervalli di tempo regolare un numero di point cloud pari alla costante `INIT_SEQ_LENGTH`. Si calcolano poi i valori di riferimento facendo la media tra i punti validi rilevati. Al termine di questa procedura il background è contenuto in `bgPcl`.

## Cambiamenti di colore e di posizione

Durante l'esecuzione del programma sono ciclicamente acquisite nuove point cloud e confrontate con il background al fine di individuare graffiti.

Per ogni punto della point cloud correntemente analizzata, contenuta in `curPcl`, si eseguono le seguenti operazioni:

1. controllo se la posizione del punto è cambiata, misurando la distanza e confrontandola con la soglia `DISTANCE_THRESHOLD`. I punti che hanno modificato posizione sono evidenziati nell'immagine `positionChange`.
2. controllo se il colore del punto è cambiato: il confronto può essere effettuato sia nello spazio dei colori RGB che in quello YUV, alternando i due metodi tramite la pressione del tasto z. Lo spazio YUV, a differenza di quello RGB, consente di distinguere cambiamenti di luminosità da quelli di tonalità. In questo modo il programma risulta più robusto in presenza di ombre sulla parete. I punti per i quali c'è stata una variazione di colore sono evidenziati nell'immagine `colorChange`

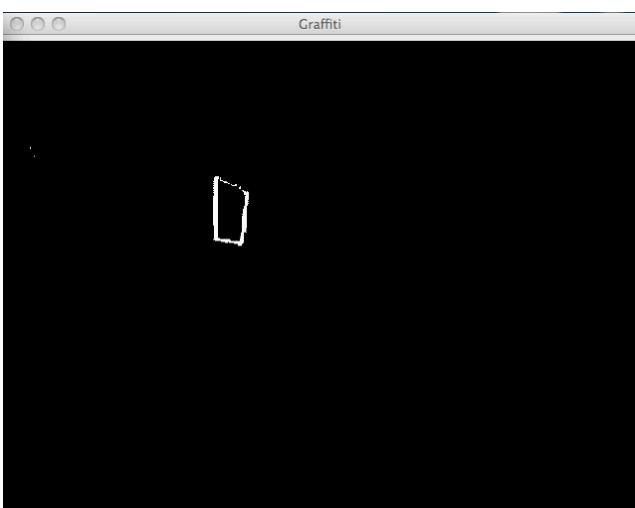
La figura seguente mostra le due immagini ottenute quando nella scena è presente un disegno e un braccio di una persona.



Si può notare il braccio della persona, oltre a causare una cambiamento di colore, provoca lo spostamento nello spazio dei punti. Il disegno invece, effettuato sulla parete dopo avere acquisito il background, provoca solamente una variazione nel colore.

Si analizzano quindi tutti i punti delle due immagini nel seguente modo:

1. se non c'è stata nessuna variazione di posizione o di colore la scena è rimasta inalterata rispetto al background.
2. se c'è stata una variazione di posizione significa che è presente un nuovo oggetto nella scena. In questo caso un'eventuale variazione di colore non è imputabile ad un atto vandalico.
3. se c'è stata una variazione di colore ma non di posizione è presente un possibile atto vandalico sul muro.



In questo un ultimo caso il programma non inoltra immediatamente una segnalazione dell'evento all'utente, poiché le misure effettuate potrebbero essere affette da rumore, ma si limita ad incrementare il contatore `changeCount` associato al punto. Un punto viene marcato come graffito solo quando questo valore supera la soglia specificata in `changeCountThreshold`, regolabile tramite l'apposito cursore nell'interfaccia grafica. Considerando che il *frame rate* è piuttosto elevato, questo non introduce significativi ritardi nella segnalazione di atti vandalici.

Al termine di questa fase l'immagine binaria `imageGraffiti` contiene tutti i punti individuati come graffiti.

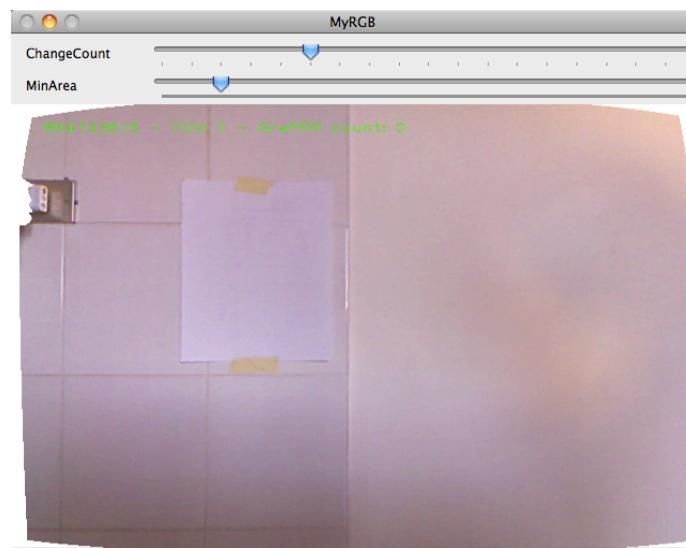
## Analisi dei graffiti

Una volta individuati i graffiti si vuole procedere ad un'analisi della dimensione e della posizione per poter inoltrare all'utente una segnalazione d'allarme. Si procede quindi al labelling delle componenti connesse presenti in imageGraffiti al fine di distinguere i diversi graffiti che possono essere presenti nella scena.

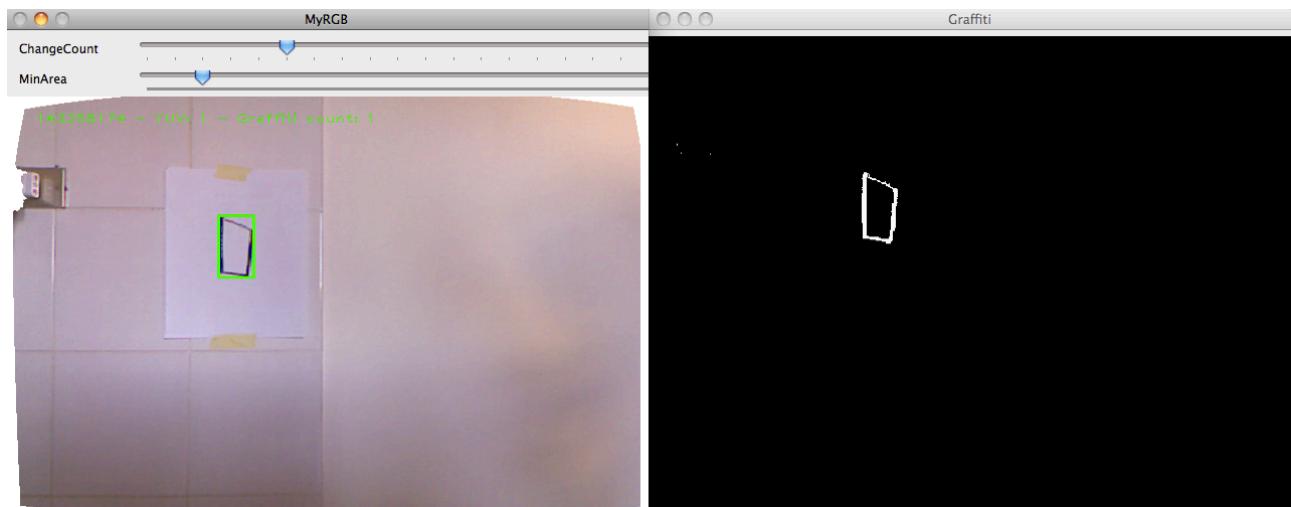
Per ognuno di questi si procede poi a determinare l'area e il rettangolo in cui è contenuto. All'utente saranno segnalati solamente i graffiti con area maggiore di areaThreshold. Anche questo valore è regolabile attraverso un cursore. Sull'immagine di output è quindi riportato il numero di graffiti individuati, evidenziati anche da un rettangolo verde.

## Risultati sperimentali

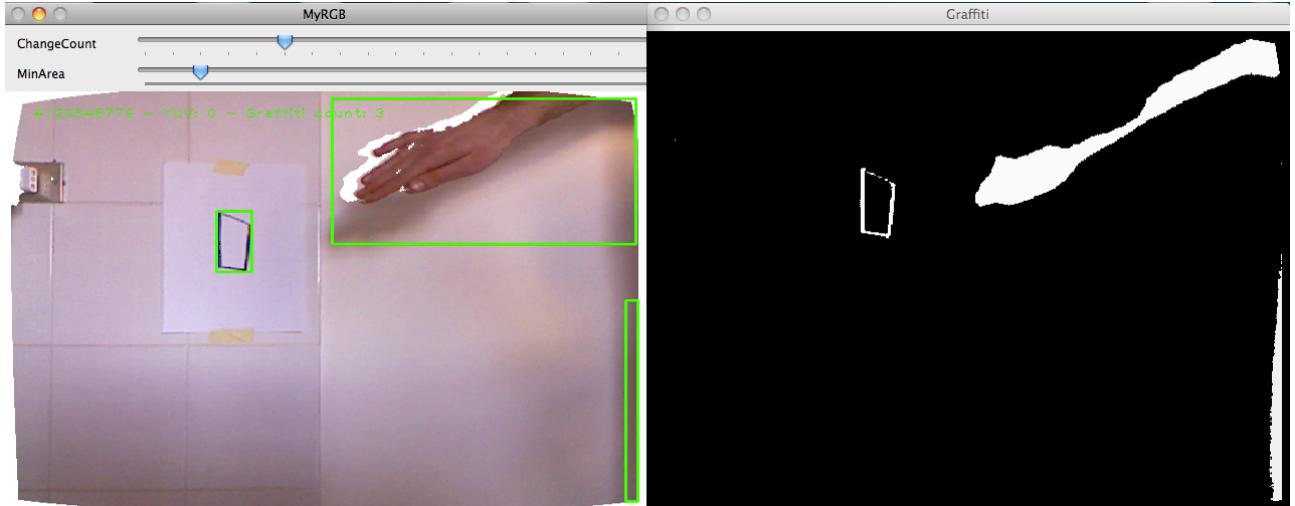
Il programma è stato testato nello scenario raffigurato nella seguente figura.



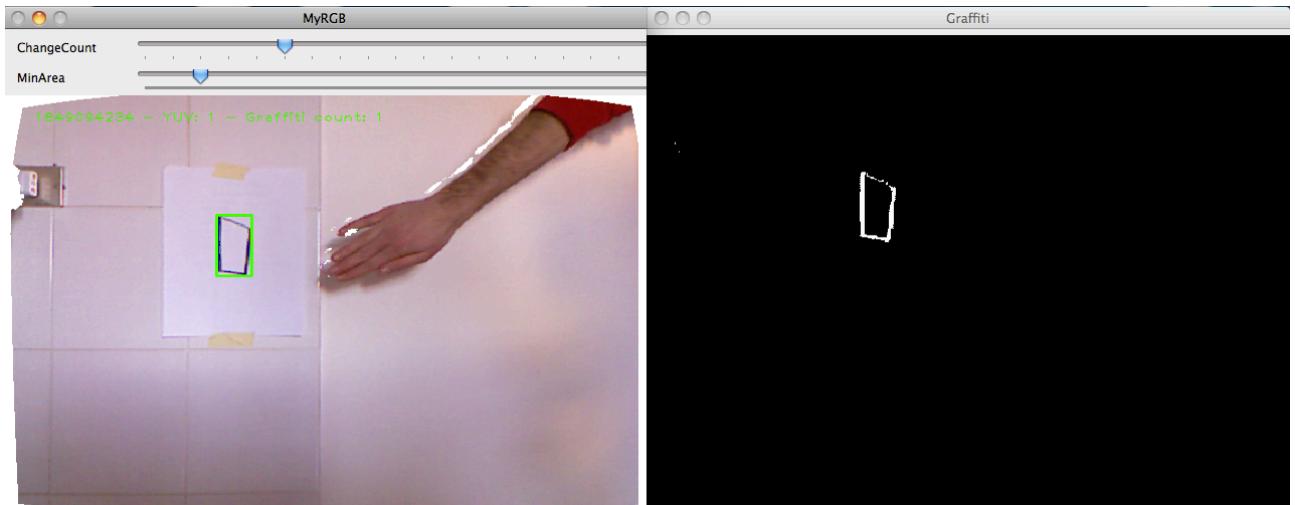
Dopo avere fatto qualche segno sul foglio con un pennarello il programma riconosce la presenza di un atto vandalico e lo segnala mostrando un rettangolo verde.



L'immagine seguente mostra come, nello spazio di colori RGB, il sistema di riconoscimento dei graffiti venga disturbato dalla presenza di ombre. Queste infatti vengono rilevate come variazioni di colore ma non di spazio e quindi assimilate a dei graffiti sulla parete.



Sono stati ottenuti significativi miglioramenti passando quindi allo spazio di colori YUV. Si suppone infatti che le ombre modifichino solamente la luminosità di un pixel senza alterare la tonalità. Impostando quindi differenti soglie per UV e Y si è in grado di eliminare numerosi falsi positivi, come mostrato nella figura seguente.



## Problemi riscontrati

I problemi principali sono legati alla determinazione di valori di soglia appropriati per discriminare cambiamenti di posizione o colore. Tali valori devono essere calibrati correttamente al fine di ridurre al minimo il numero di falsi positivi o falsi negativi.

# HAND GESTURE

Questo programma individua la mano dell'utente e riconosce alcune semplici gesture, spostando quindi il cursore del mouse del computer. Nei paragrafi seguenti saranno descritte le operazioni necessarie per estrarre la forma della mano e analizzarne l'orientamento.

## Estrazione del braccio

Il programma è in grado di rilevare la mano se essa si trova in primo piano: la mano deve essere l'oggetto più vicino al Kinect. In questo modo è possibile effettuare una prima scrematura dei punti che si trovano ad una distanza superiore a quella della mano, riducendo notevolmente il carico computazionale per le operazioni successive.

Poiché si vogliono individuare anche gesture che prevedono la rotazione della mano è opportuno mantenere tutti i punti all'interno di una certo intervallo di distanze di larghezza pari a `DISTANCE_THRESHOLD` - per ora fissato a 20cm. La funzione `pclDistThreshold` invalida tutti i punti che non sono compresi all'interno di tale intervallo.

Al termine di questa operazione, gli unici punti validi saranno quelli della mano e quelli dell'avambraccio.

## Determinazione del colore della mano

Il colore della pelle della mano è un parametro che non può essere definito in modo univoco. Questo infatti cambia da persona a persona e inoltre risente notevolmente delle condizioni di illuminazione della scena. L'immagine mostra alcuni colori tipici della pelle di persone appartenenti a diverse etnie.



Dal momento che si vuole rendere indipendente il programma rispetto al colore della pelle, vi è la necessità di stimare quale sia il colore della pelle specifico dell'utilizzatore. Per farlo si è sviluppata un'apposita libreria `libSkin`.

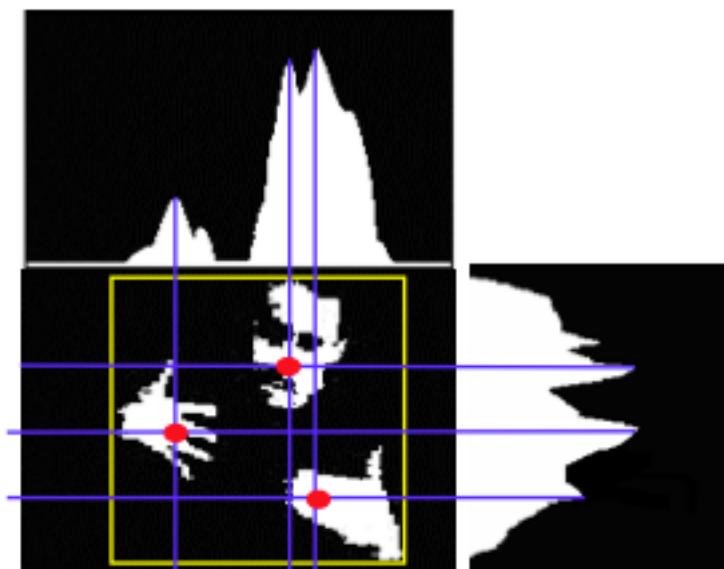
Le funzioni proposte sono state sviluppate prendendo spunto dalla tecnica illustrata nell'articolo "Vision-Based Skin-Colour Segmentation Of Moving Hands For Real-Time Applications" di S. Askar e altri.

In prima istanza il programma utilizza un "colore di pelle globale" per cercare di individuare un punto che con buona probabilità appartiene alla mano. Il colore di pelle globale è stato fissato come segue, effettuando una media di tutti i possibili valori:

```
#define GLOBAL_SKIN_BLUE 207
#define GLOBAL_SKIN_GREEN 208
#define GLOBAL_SKIN_RED 239
```

Utilizzando una soglia molto abbondante (GLOBAL\_SKIN\_COLOR\_THRESHOLD) e ricorrendo alla distanza euclidea tra i colori, si può quindi effettuare una prima binarizzazione dell'immagine nella quale vengono evidenziati i punti potenzialmente appartenenti alla mano.

Costruendo poi due histogrammi e individuando i punti di massimo è possibile individuare dei punti posizionati al centro della mano. L'immagine tratta dal paper mostra l'applicazione di questo procedimento per determinare la posizione delle due mani e del viso, considerando quindi 3 punti di massimo relativo.

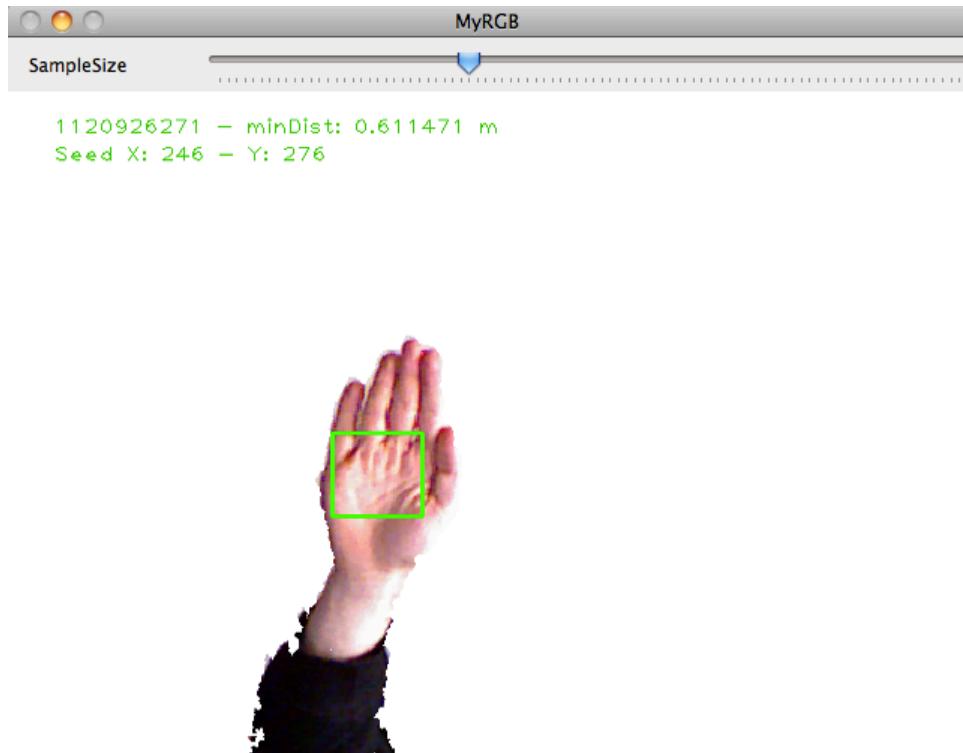


In questo programma, poiché è necessario individuare solo una mano, è sufficiente considerare solo il punto di massimo assoluto. Tale operazione è svolta dalla seguente funzione

```
void getSkinSeed(PointCloud pcl, int* seedSkinX, int* seedSkinY);
```

Data una point cloud, restituisce le coordinate del seed point utilizzato in seguito.

La figura seguente mostra il risultato delle prime due fasi di elaborazione. La point cloud contiene solamente l'avambraccio e la mano, mentre il seed point individuato è posizionato al centro del rettangolo verde, nelle coordinate (264, 276).



Dopo aver individuato un punto della mano si deve procedere all'acquisizione di un campione di punti della pelle. Questa fase è fondamentale poiché, dall'analisi dei punti raccolti, il programma provvede a creare un modello del colore della pelle dell'utilizzatore che sarà utilizzato in seguito per individuare la mano.

Poiché l'operazione deve essere svolta con una certa precisione viene mostrato all'utente un rettangolo centrato nel seed point e contenente tutti i punti che saranno utilizzati come campione. La dimensione del campione (e quindi del rettangolo) è data dalla variabile `sampleSize` e può essere modificata a piacimento con un cursore.

Quando l'utente ha centrato il rettangolo sulla propria mano in modo da coprirne buona parte può procedere all'acquisire un campione di punti, premendo il tasto x. Tali punti verranno inseriti nel vettore `sampleVec` e poi analizzati come descritto nel seguito. L'utente può anche acquisire più campioni in momenti successivi, premendo nuovamente il tasto x. In questo caso i nuovi punti vengono aggiunti a quelli già esistenti e viene svolta nuovamente l'analisi.

Una volta acquisiti i punti si cerca di stabilire quale sia la distribuzione della pelle dell'utilizzatore nello spazio dei colori YUV. Per ottenere una maggior precisione infatti si misurerà la distanza tra due colori utilizzando la distanza di Mahalanobis al posto di quella euclidea.

Si calcola quindi il valore medio (`meanColorVec[3]`) e la matrice di covarianza (`colorCovarMat[3][3]`) del colore di tutti i punti del campione acquisito attraverso la funzione

```
int grabSkinSample(PointCloud pcl, int seedSkinX, int seedSkinY, int sampleSize);
```

Di seguito è riportato l'output della console ottenuto durante l'esecuzione del programma.

```
SampleSize = 3721  
Y = 0.746958 U = -0.038477 V = 0.145655
```

```
===== Covar =====  
C[0][0]= 0.024928 C[0][1]= 0.000619 C[0][2]= -0.010037  
C[1][0]= 0.000619 C[1][1]= 0.000832 C[1][2]= -0.000720  
C[2][0]= -0.010037 C[2][1]= -0.000720 C[2][2]= 0.005869  
=====
```

```
===== CovarInv =====  
C[0][0]= 137.713864 C[0][1]= 113.130816 C[0][2]= 249.375674  
C[1][0]= 113.130816 C[1][1]= 1436.688356 C[1][2]= 551.546132  
C[2][0]= 249.375674 C[2][1]= 369.618061 C[2][2]= 642.158101  
=====
```

Si noti in particolare che nello spazio di colori YUV, il colore della pelle tende ad essere concentrato in una zona molto ristretta del piano UV, mentre la varianza di Y è più elevata.

I valori calcolati saranno utilizzati nei passaggi successivi per estrarre la forma esatta della mano, distinguendola da eventuali parti dell'avambraccio ancora presenti nella point cloud. Si suppone quindi che l'utilizzatore stia utilizzando una maglia a maniche lunghe di colore significativamente diverso dalla propria pelle. In caso contrario il programma non sarebbe in grado distinguere la mano dal resto dell'avambraccio.

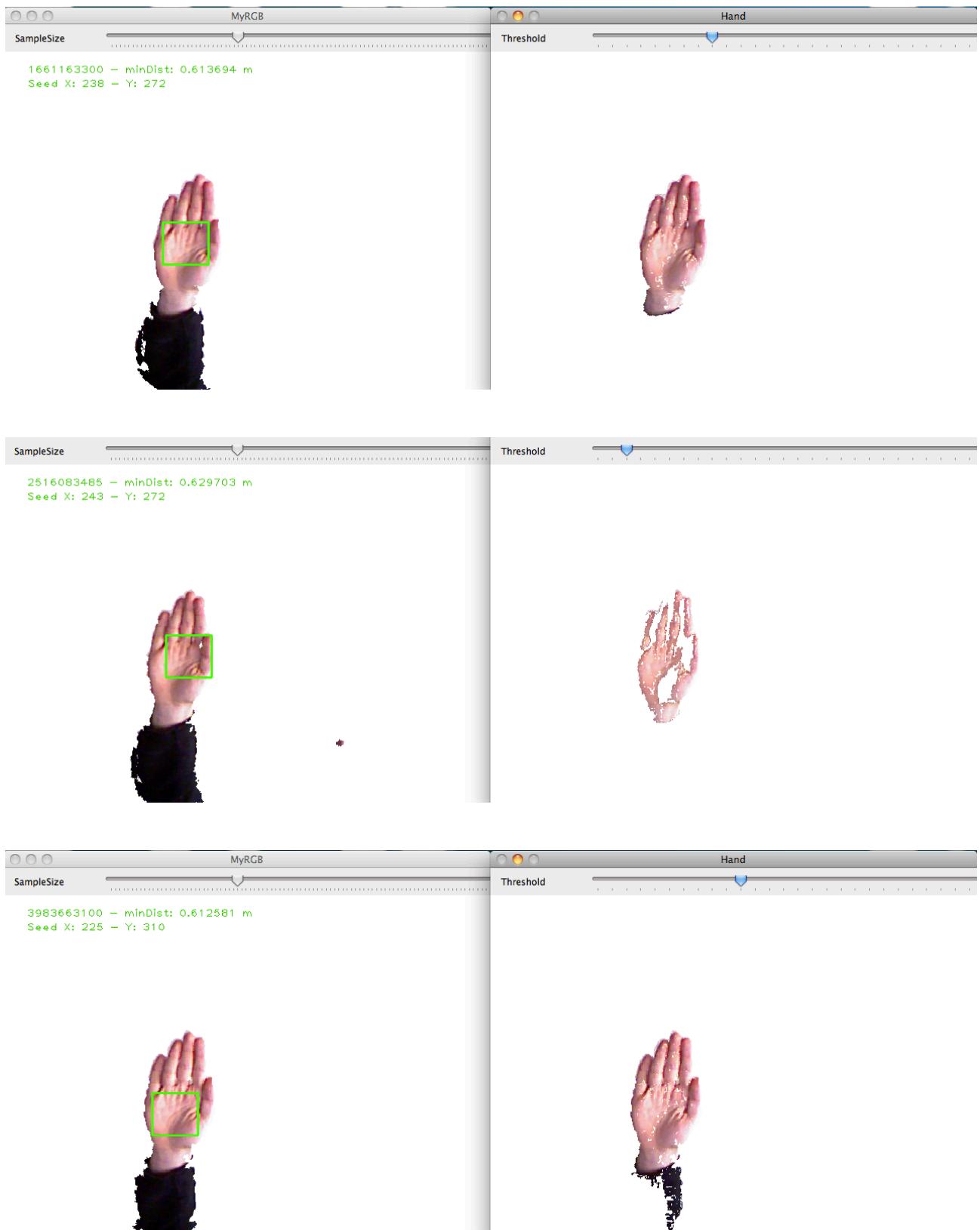
## Individuazione della mano

Questa operazione viene avviata automaticamente dopo l'acquisizione del primo campione di punti. A partire dal seed point individuato nel frame corrente, si esegue un algoritmo di region growing con lo scopo di individuare tutti i punti appartenenti alla mano.

L'operazione è effettuata dalla funzione

```
void regionGrowing(PointCloud src, PointCloud dst, int threshold)
```

I punti vengono annessi alla mano solo se la differenza di colore misurata attraverso la distanza di Mahalanobis è inferiore alla soglia specificata `specSkinColorThreshold`. Questo parametro deve essere regolato in modo opportuno tramite il cursore, al fine di ottenere una point cloud priva di punti dell'avambraccio, ma allo stesso tempo senza troppi buchi. Nella prima immagine il livello della soglia è corretto e la mano viene individuata correttamente. La seconda e la terza immagine mostrano i risultati ottenuti rispettivamente che soglia troppo bassa e soglia troppo alta.



La distanza di Mahalanobis è calcolata tramite la seguente funzione  
`float colorMahalanobisDistance(Point3d point, double meanColorVec[3], double inverseCovarMat[3][3])`

alla quale vengono quindi passati di volta in volta i punti da confrontare con il valore medio del campione.

## Analisi della posizione e dell'orientamento

Le funzioni contenute nel file gestureManager si occupano di riconoscere e interpretare la posizione e l'orientamento della mano. Ogni volta che viene individuata una nuova point cloud della mano viene invocato il metodo updateHandPosition.

La posizione della mano nello spazio è modellata come un punto nello spazio, posizionato nel baricentro della point cloud.

In questo contesto si suppone poi che il palmo della mano sia aperto, in modo che tutti i punti siano disposti sullo stesso piano. Per determinare l'orientazione di tale piano si è utilizzato il metodo della matrice di covarianza delle coordinate spaziali. Gli autovettori di questa matrice rappresentano infatti l'asse maggiore, l'asse minore e il vettore normale alla mano.

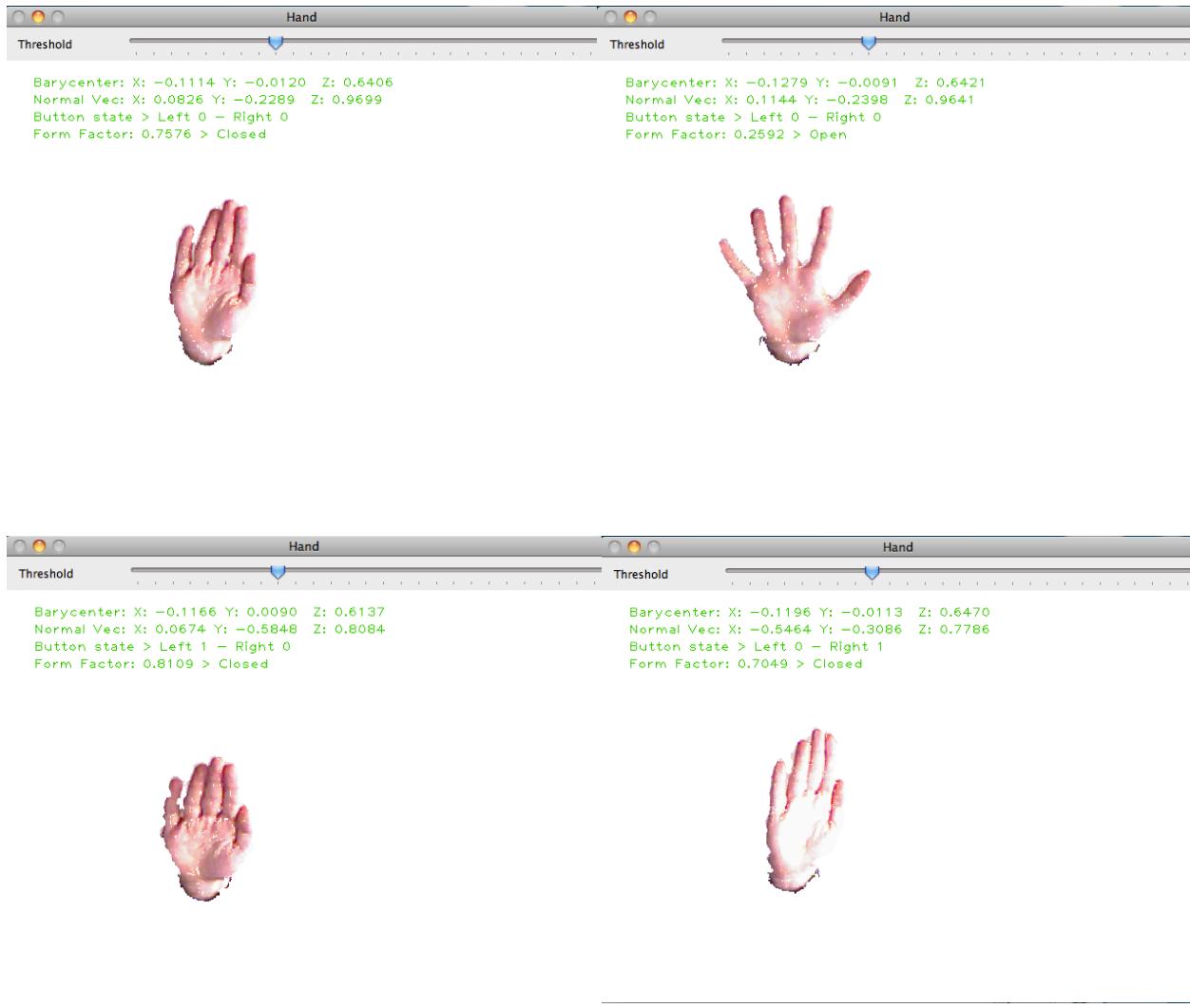
Il metodo openHand determina infine se la mano è aperta o chiusa, calcolando il fattore di forma dell'immagine 2D e confrontandolo con un valore soglia OPEN\_HAND\_FORM\_FACTOR.

A questo punto si può procedere a muovere il mouse in modo coerente con la posizione della mano appena calcolata. Il metodo moveMouse interpreta le gesture possibili e esegue le azioni opportune. Il programma supporta le seguenti gesture:

- Movimento della mano chiusa: spostamento del mouse
- Movimento verticale della mano aperta: scrolling verticale (equivalente alla rotellina del mouse)
- Flessione in avanti della mano: click sinistro
- Rotazione della mano: click destro

Le immagini seguenti mostrano le gesture interpretate dal programma. Le scritte in alto a sinistra in ciascuna immagine indicano rispettivamente:

- Coordinate del baricentro
- Direzione del versore normale al piano della mano
- Stato dei bottoni sinistro e destro del mouse, dove 0 indica che il pulsante è rilasciato mentre 1 che il pulsante è premuto
- Fattore di forma della mano, dal quale è possibile capire se la mano è aperta o chiusa



Il mouse è controllato tramite la libreria `libMouse OSX`, unica parte platform dependent del software, realizzata nello specifico per il sistema operativo Mac OSX. Resta comunque facilmente implementabile una libreria analoga anche per altri sistemi operativi.

## Risultati sperimentali

Dopo un breve periodo di training da parte dell'utente, che deve l'applicazione permette di controllare il mouse come una discreta precisione anche se non permette di cliccare su elementi di dimensioni troppo piccole. L'utilizzo è da considerarsi quindi limitato ad applicazioni con interfaccia grafica molto semplice.

Per testare il software è stato quindi sviluppato un programma d'esempio (PlayGround) contenente alcuni pulsanti di grandi dimensioni e un'area di testo nel quale è possibile effettuare lo scroll.

Le performance in termini di velocità di esecuzione sono soddisfacenti, anche se si potrebbero incrementare ulteriormente effettuando alcune ottimizzazioni al software che è stato scritto con lo scopo di essere facilmente comprensibile piuttosto che efficiente.

## **Problemi riscontrati**

I problemi principali di questo software sono legati al riconoscimento del colore della pelle. Anche se l'approccio basato su colore globale e colore specifico ha dato buoni risultati restano ancora evidenti problematiche. In primo luogo il colore globale della pelle non può essere stabilito in modo preciso, ma è un valore stabilito in modo empirico. Inoltre il colore rilevato dal sensore è fortemente influenzato dall'illuminazione della stanza: muovendo e ruotando la mano il colore della pelle può cambiare notevolmente se la sorgente luminosa si trova in determinate posizioni.

Esistono poi alcune limitazioni per l'utilizzatore, il quale deve indossare una maglia a maniche lunghe e di colore diverso dalla pelle.

# CONCLUSIONI

Grazie alla tecnologia del Kinect sono state realizzate due applicazioni che, con l'utilizzo di una semplice webcam, sarebbero state difficilmente implementabili. Le informazioni sulla profondità fornite direttamente dall'hardware permettono l'impiego di questo dispositivo in numerosi contesti applicativi tanto che in rete stanno nascendo numerosi progetti di ricerca e non solo.

Le due applicazioni sono state testate con successo e hanno mostrato buone performance anche se sono ancora presenti alcuni problemi dovuti alla calibrazione dei parametri del sistema.

# APPENDICE

## Compilazione del software

Il software richiede la libreria OpenCV per l'elaborazione delle immagini, oltre alla libreria libfreenect per interfacciarsi al dispositivo.

Per procedere alla compilazione è necessario l'utilizzo del software cmake.  
Scaricare quindi il sorgente dal repository <https://github.com/Muffo/KinectGraffiti>

Aprire quindi un terminale e posizionarsi nella cartella principale del progetto. Digitare quindi i seguenti comandi:

```
mkdir build  
cd build  
cmake ../src  
make
```

In questo modo si compileranno le due applicazioni e saranno disponibili i rispettivi file eseguibili chiamati “graffiti” e “hand”.

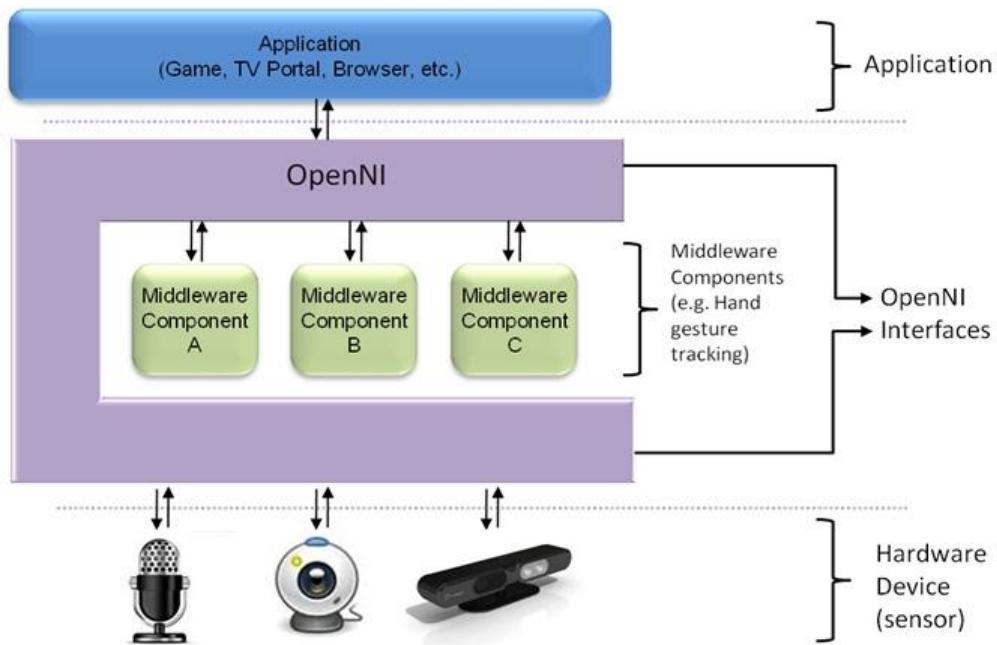
Se si preferisce utilizzare un IDE è possibile generare automaticamente un progetto digitando i comandi

```
cmake -G Xcode ../src  
oppure  
cmake -G Eclipse CDT4 – Unix Makefiles ../src
```

## OpenNI

Visto il successo del dispositivo anche al di fuori dell'ambito videoludico dell'Xbox 360, PrimeSense è in procinto di rilasciare dei driver ufficiali per il Kinect anche per PC. In questo progetto non è stato possibile utilizzare OpenNI poiché non era ancora disponibile una release pubblica.

Il progetto OpenNI, permetterà di accedere alle piene funzionalità del dispositivo e fornirà anche un middleware per la scrittura di software ad un livello più elevato. Come mostrato nella seguente figura sarà possibile accedere sia alle informazioni grezze provenienti dai sensori che a informazioni di più alto livello, come il tracking del corpo delle persone.



Maggiori informazioni sono disponibili sulla pagina ufficiale del progetto: <http://www.openni.org>