Christian Onzaca (N46-001351)
Antonio Pagano (N46-001239)
Daniele Vergara (N46-001562)

Tesina di Sistemi Multimediali

Sistema di videosorveglianza in Open-CV

Facoltà di Ingegneria Informatica
Università Federico II di Napoli

2015

Indice

Capitolo 1

Introduzione

Al giorno d'oggi, la tecnologia è sempre più importante nella vita dell'uomo. Sorge l'esigenza di realizzare sistemi informatici che emulino la percezione umana della realtà. Nasce così la Computer Vision, ossia una scienza teorica informatica nata come branca dell'Intelligenza Artificiale, che cerca di emulare gli aspetti e le caratteristiche della visione umana per i computer. Le applicazioni che ne fanno uso sono molteplici: dal riconoscimento facciale a quello di oggetti, quindi all'object tracking, fino ad arrivare alla rilevazione di azioni sospette. Nascono così i moderni sistemi di videosorveglianza, adoperati sempre più spesso per la sicurezza di banche, aziende, case etc.

Il nostro obiettivo è proprio quello di implementare le basi per il funzionamento di un moderno sistema di videosorveglianza, utilizzando le librerie OpenCV e l'ambiente di sviluppo Microsoft Visual Studio.

La finalità del progetto saranno:

- Individuazione di figure in movimento;
- Riproduzione di un suono di allarme in caso di rilevamento;
- Accesso protetto al software;
- Acquisizione di immagini in caso di rilevamento.

Capitolo 2

Strumenti Utilizzati

2.1 Open-CV

È un insieme di librerie open source, scritte in C, il cui obiettivo é fornire delle funzionalitá relative alla Computer Vision. È compilabile su Windows, Linux e Mac OS X, ed utilizzabile con C/C+, Java e Python.

2.2 Microsoft Visual Studio Express

È un ambiente di sviluppo Microsoft integrato, basato sulla piattaforma .NET Framework, che supporta più linguaggi di programmazione (nel caso specifico, abbiamo utilizzato il C++). Fornisce anche un'interfaccia semplice ed intuitiva per la creazione, modifica e gestione dei Windows Forms.

Capitolo 3

La videosorveglianza

Un sistema di videosorveglianza é generalmente composto da quattro layer (strati):

1. Sensor Layer: si occupa dell'acquisizione delle informazioni sull'ambiente osservato.

2. Image Processing Layer: estrae le informazioni raccolte dal Sensor Layer e li elabora attraverso tecniche di Image Processing applicate da un modulo chiamato IPS (Image Processing System.

3. Composite Event Detection Layer: rileva e riconosce eventi complessi ad alto livello a partire da eventi a basso livello restituiti dall'IPS (ad esempio, frugare in un cassetto).

4. User Layer: permette la gestione del sistema da parte dell'utente.

Nel nostro caso, si é implementato un sistema di videosorveglianza con i seguenti layer:

1. Si è utilizzato come Sensor Layer una semplice webcam (hardware a basso costo). Ció implica che il flusso di dati fornito ai livelli successivi potrebbe essere di bassa qualitÃ .

2. L'IPS implementato utilizza un algoritmo di segmentazione degli oggetti basato su Frame Difference, cioè sono evidenziati gli oggetti che presentano delle differenze di intensità apprezzabili tra due frame successive (indicazioni sull'algoritmo per movement detection).

3. Il sistema di videosorveglianza in esame non presenta il livello 3.

4. L'utente può personalizzare le impostazioni del programma (ad esempio, avere la possibilità di acquisire shots con una determinata frequenza) tramite una GUI.

Capitolo 4

Image Processing

Prima di essere in grado di rilevare accuratamente il movimento, i flussi di frame devono essere elaborati opportunamente in tempo reale. Risultano, quindi, necessarie tecniche di Image Processing ad-hoc, ossia trattamenti dell'immagine necessari per adeguarla allo scopo. A tal fine, verranno eseguite operazioni di filtraggio e manipolazione. Le tecniche utilizzate, sono:

- Smoothing Spatial Filter (medio)
- Splitting
- Thresholding

Per filtro di tipo spaziale si intende un tipo di filtraggio che viene applicato a ogni pixel e al suo neighborhood al fine di ottenere nuovi pixels. A tale scopo vengono adoperate strutture dette Maschere. Esse sono matrici di pixel con valori opportuni, e verranno utilizzate per operazioni algebriche su ogni singolo pixel dell'immagine da filtrare.

4.1 Smoothing Spatial Filter (Blurring)

Il filtro di smoothing è un un filtro di tipo spaziale che effettua un addolcimento dell'immagine rimuovendo i dettagli e omogeneizzando le sue regioni. Con questa operazione vengono enfatizzate le zone a maggior variazione di colore, ponendo in secondo piano i dettagli poco utili. Il filtro di smoothing medio è un filtro lineare, non statistico e general purpose, che assegna ad ogni pixel dell'immagine un valore pari alla media della sua convoluzione con la maschera, cioè si effettua la seguente operazione:

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+s,y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)}$$

con

$$a = \frac{m-1}{2} \qquad b = \frac{n-1}{2}$$

dove w è la maschera di dimensioni mxn, f(x, y) è il generico pixel dell'immagine in ingresso e g(x, y) è il corrispondente pixel dell'immagine in uscita. Tale operazione riduce l'effetto del rumore, spalmando eventuali livelli alti di grigio sull'immagine.

4.2 Splitting

Lo splitting è un'operazione che consiste nella divisione di un singolo frame nei relativi canali R, G e B. Dunque, si avrà in uscita un insieme di tre frames in scala di grigio, ciascuno dei quali rappresentativo dei livelli di rosso, verde e blu nell'immagine di partenza. Questa operazione é fortemente necessaria per poter applicare il metodo del thresholding su ogni canale.

4.3 Thresholding

Il Thresholding è un metodo che, dato in ingresso un'immagine in scala di grigi, restituisce una seconda immagine binaria (più precisamente in bianco e nero). Questa immagine è costruita ponendo un valore di soglia S al di sotto del quale i pixel vengono settati come bianchi, neri altrimenti:

$$p_{i,j} = \begin{cases} 0 & t_{i,j} \leq S \\ 255 & t_{i,j} > S \end{cases}$$

con $p_{i,j}$ generico pixel dell'immagine in uscita e $t_{i,j}$ generico pixel dell'immagine in entrata. Il thresholding è principalmente utilizzato per riconoscere marcatamente i punti in cui l'immagine presenta dei salti (cioè passaggi da bianco a nero). In questo modo vengono ben definiti i contorni (edge). Di contro tale tecnica può produrre del rumore fornendo all'immagine il caratteristico effetto salt and pepper.

Capitolo 5

Algoritmo di rilevamento dell'intruso (IntruderAlarm)

È l'algoritmo centrale del progetto. È una funzione booleana che restituisce vero se viene individuato un corpo in movimento. L'intestazione è la seguente:

bool *intruderAlarm*(cv::Mat& **a**, cv::Mat& **b**)

dove **a** e **b** rappresentano due frame successive acquisite dalla webcam:

La prima operazione da eseguire è sfocare le due frame per poter eliminare dettagli inessenziali e individuare solo figure in primo piano.

Quindi, si utilizza il metodo *blur*, la cui intestazione è:

void *blur*(InputArray **src**, OutputArray **dst**, Size **ksize**)

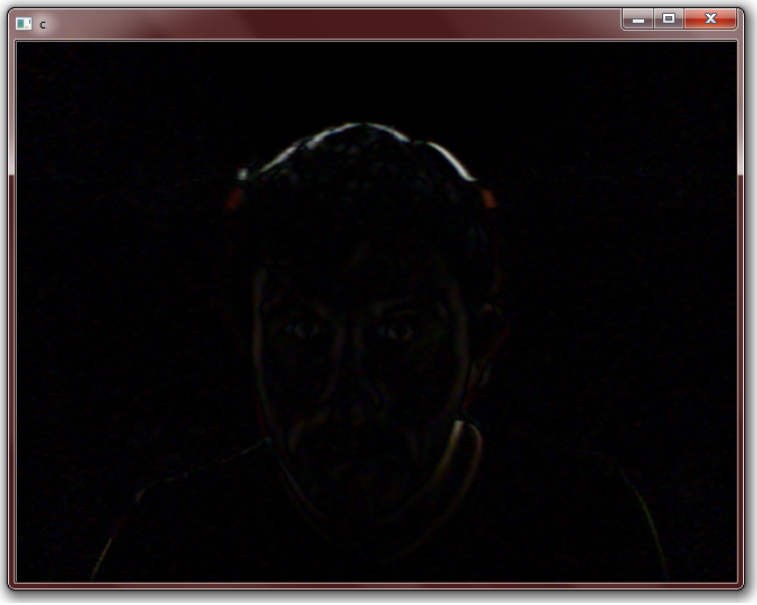
dove **src** è la frame in ingresso (dunque, prima a e poi b), **dst** è la frame in uscita (rispettivamente, a_blurred e b_blurred) e **ksize** indica le dimensioni della maschera con cui fare la convoluzione. Più grandi saranno le dimensioni di questa maschera, più l'immagine sarÃ sfocata. Si è scelta una maschera quadrata di dimensioni 3x3, in modo da coinvolgere ogni singolo pixel e il relativo vicinato.
Ecco le due frame a_blurred e b_blurred:

Successivamente, è necessario confrontare le due frame acquisite e sfocate. A tal scopo, viene utilizzato il metodo *absdiff*:

void *absdiff*(InputArray **src1**, InputArray **src2**, OutputArray **dst**)

dove **src1** è il primo frame (quindi, a_blurred), **src2** è il secondo frame (b_blurred) e **dst** il frame in uscita (c). Nella frame c verranno evidenziate principalmente le zone in cui si hanno maggiori variazioni di colore, ossia quelle in cui vi è un oggetto in movimento. Quindi, lo sfondo, che è statico, apparirà nero (non vi sono differenze di colore tra le due frame), mentre

eventuali figure in primo piano che si muovono verranno evidenziate (qui vi saranno differenze di colore apprezzabili). Ecco la frame **c**:



Ora, per poter rendere più efficiente e rapido il rilevamento di oggetti in movimento, occorre basarsi su immagini più semplici e dal contenuto informatico più denso: si può pensare, infatti, di lavorare su un'immagine binaria, ossia in bianco e nero, per rendere più leggere le successive elaborazioni. Prima, però, occorrerebbe dividere l'immagine nei relativi canali RGB (rosso, verde e blu), in modo da poter lavorare sulle relative scale di grigio. A tal scopo, si utilizza il metodo *split*:

void *split*(const Mat& **src**, Mat* **mvbegin**)

dove **src** è la frame in ingresso (quindi, c) e **mvbegin** è un vettore di frame in uscita (chiamato **channels**). Le frame del vettore **channels**, adesso, risulteranno essere in scala di grigi: ciò renderà più semplici le elaborazioni successive.

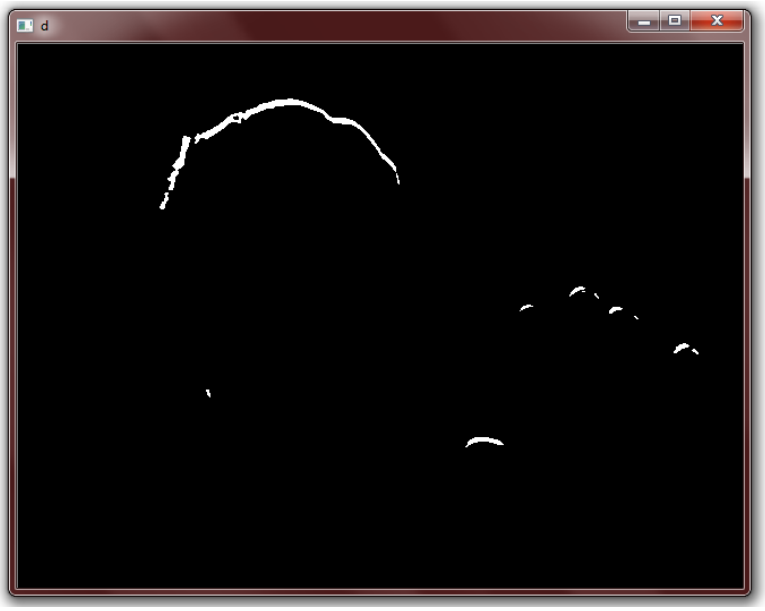
Le frame contenute nel vettore **channels** saranno le seguenti:

In seguito, si prende una matrice d di zeri e, per ogni canale si costruisce una frame che presenta pixel bianchi al di sopra di una certa soglia, e pixel neri al di sotto della stessa.

Quindi, si utilizza su ogni frame del vettore channels il metodo *threshold*:
double *threshold*(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**)

dove **src** è la frame in ingresso (channels[i]), **dst** è la frame in uscita (viene creata una frame thresh, che ad ogni ciclo viene messa in OR con un'altra frame d), **thresh** è il valore di soglia (si è scelto arbitrariamente un valore di soglia basso, thresh = 45), **maxval** è il valore che viene assegnato al pixel nel caso in cui esso presenti un valore maggiore di quello di soglia (qui si utilizza il bianco, quindi maxval = 255) e, infine, **type** è il tipo di thresholding utilizzato (si è impiegato quello binario, quindi type = CV_THRESH_BINARY).

C'è una ragione precisa per cui si effettua questa operazione: infatti, dopo aver ottenuto la differenza tra le due frame, e averne individuato i canali, gli sfondi risulteranno essere molto scuri, mentre i contorni degli oggetti in movimento risulteranno essere più chiari. Effettuando il thresholding binario, dunque, lo sfondo e gli oggetti diventeranno neri, mentre i contorni degli oggetti in movimento diventeranno bianchi.
Queste considerazioni valgono per ciascuna delle frame del vettore channels. Dunque, di ogni frame thresh in uscita dall'operazione di thresholding, si effettua la OR con la frame d: in questo modo, in d saranno presenti insieme tutte le zone evidenziate dal thresholding di ogni singola frame del vettore channels.
Questo è il risultato di queste operazioni, ossia la frame d:



Per rilevare un oggetto in movimento, bisogna comunque individuarne la sagoma. Per cui, data un'immagine di cui vengano evidenziati i contorni, la sagoma si evidenzia riempiendo l'interno di questi ultimi. Per fare ciò, ovviamente, tali contorni dovranno essere figure chiuse. Tuttavia, prima di procedere con questa operazione, è necessario un elemento strutturale, il quale rappresenterà l'unità base per questa operazione.

Per ottenerlo, si utilizza la funzione *getStructuringElement*:

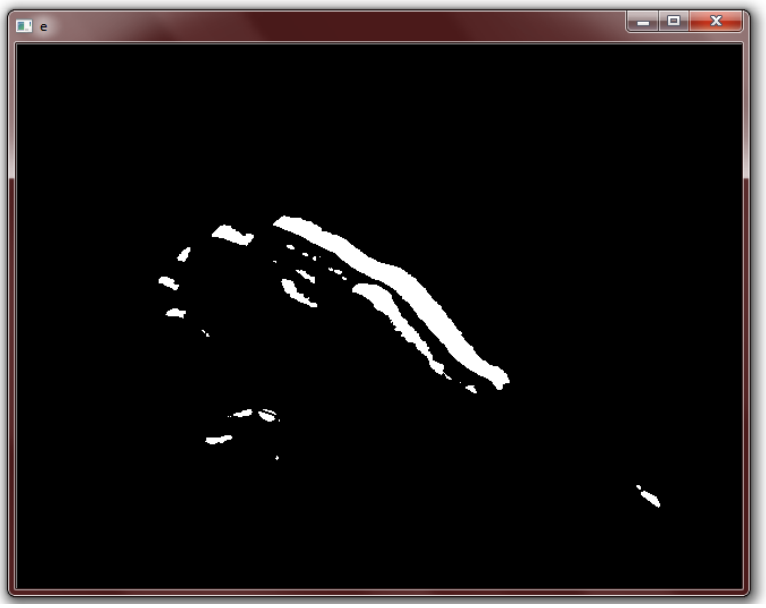
Mat *getStructuringElement*(int **shape**, Size **ksize**)

dove **shape** è il tipo di figura geometrica che si vuole utilizzare (si è scelto MORPH_RECT, cioè un rettangolo) e **ksize** indica le dimensioni di tale figura (si è scelto Size(1, 1)). A questo punto, si può procedere al riempimento della figura all'interno dei contorni, utilizzando l'elemento strutturale appena creato, il quale viene inserito in una matrice chiamata kernel.

Quindi, si richiede di eseguire un'elaborazione non su un'immagine, ma sulla forma di un oggetto presente nella stessa, quindi un'operazione morfologica. A tal fine, si utilizza il metodo *morphologyEx*:

void *morphologyEx*(InputArray **src**, OutputArray **dst**, int **op**, InputArray **kernel**, Point **anchor**, int **iterations**)

dove **src** è la frame in ingresso (si è utilizzata la frame d), **dst** è la frame in uscita (si è utilizzata una nuova frame e), **op** indica il tipo di operazione morfologica da eseguire (in questo caso, op = MORPH_CLOSE, in quanto dobbiamo riempire l'interno di contorni che formano una figura chiusa, eliminando eventuale rumore sotto forma di pixel neri), **kernel** è l'elemento strutturale (kernel, appunto), **anchor** è il punto di riferimento da cui si deve partire per applicare il nostro elemento strutturale (il punto (-1, -1)), e **iterations** indica il numero di cicli in cui questa operazione deve essere ripetuta (un buon risultato si ottiene iterando il procedimento per ben 5 volte).
La frame e che si ottiene è la seguente:



Adesso, bisogna recuperare i contorni dalla frame e elaborata tramite l'-operazione morfologica. A tal fine, è necessaria un vettore di vettori di punti (chiamata contours), nella quale tali contorni verranno inseriti. Un contorno è visto praticamente come un vettore di punti.

È possibile inserire i contorni in contours attraverso il metodo *findContours*:

void *findContours*(InputOutputArray **image**, OutputArrayOfArrays **contours**, int **mode**, int **method**)

dove **image** è la frame in ingresso (si è utilizzata la frame e), **contours** è il vettore di vettori di punti in cui verranno inseriti i contorni (contours, appunto), **mode** indica la modalitÃ di riconoscimento dei contorni (si è utilizzata la modalitÃ CV_RETR_EXTERNAL, la quale riconosce le parti più esterne delle figure in movimento), e **method** indica il metodo di approssimazione dei contorni (si è scelto CV_CHAIN_APPROX_SIMPLE, il quale approssima il contorno con un insieme di segmenti, dei quali vengono memorizzati solo gli estremi).

Arrivati a questo punto, bisogna verificare effettivamente se ci sono oggetti in movimento nelle frame catturate dalla webcam. A tal scopo, si utilizza un altro vettore di vettori di punti, chiamato **intruders**. Per poter procedere, bisogna determinare l'area delle figure all'interno di ciascuno dei contorni contenuti nel vettore contours. Per fare ciò, si utilizza la funzione *contourArea*:

double *contourArea*(InputArray **contour**)

che restituisce l'area del contorno in ingresso: **contour** indica il contorno di cui calcolare l'area. Questa operazione va fatta per ogni contorno presente nel vettore contours. Se l'area dell'i-esimo contorno di questo vettore è maggiore o uguale ad una certa quantità, allora tale contorno viene inserito nel vettore di vettori di punti intruders. Dunque, per capire se è stato rilevato un movimento, basterà osservare la dimensione del vettore intruders: se essa è maggiore di zero, allora un nuovo movimento è stato rilevato.
Per far osservare all'utente l'oggetto in movimento, è necessario applicare una maschera mask alla frame b. Tale maschera avrà il compito di lasciare inalterata la parte all'interno dei contorni (evidenziati) delle figure rilevate, e di oscurare la parte esterna. Questa operazione è possibile grazie all'utilizzo del metodo *drawContours*:

void *drawContours*(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int thickness)

dove **image** è la frame in cui mettere i contorni (nel caso in analisi, è la frame mask), **contours** è il vettore dei contorni (quindi contours), **contourIdx** è un identificatore del contorno da considerare (si è posto contourIdx = -1 in quanto un valore negativo assicura che tutti i contorni siano presi), **color** è il colore utilizzato per i contorni (si è scelto ovviamente il bianco), e **thickness** è lo spessore con cui i contorni verranno evidenziati (anche qui, scegliendo thickness = -1, si avrà una situazione particolare, ossia quella in cui la parte interna dei contorni verrà completamente riempita).

A questo punto, si può procedere all'applicazione della maschera. Si osservi innanzitutto che:

- dividendo i livelli RGB associati ad un singolo pixel per un valore intero maggiore di uno, la frame risultante sarà oscurata;

- applicando l'operatore ˜ (tilde) ad una frame, si ottiene la corrispondente frame complementare (ossia, nel caso di una frame in bianco e nero, le zone nere diventeranno bianche e quelle bianche diventeranno nere);

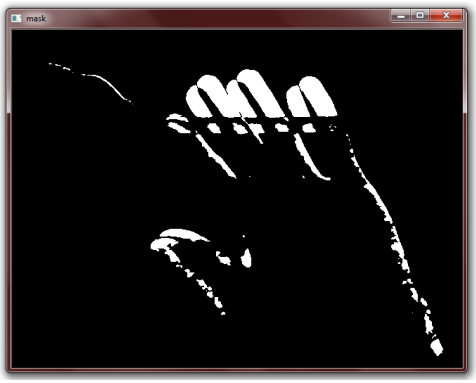
- la AND tra i pixel di una frame a colori e quelli di una frame in bianco e nero darà il pixel della prima frame nelle regioni corrispondenti alle zone bianche della seconda frame, pixel neri nelle regioni rimanenti.

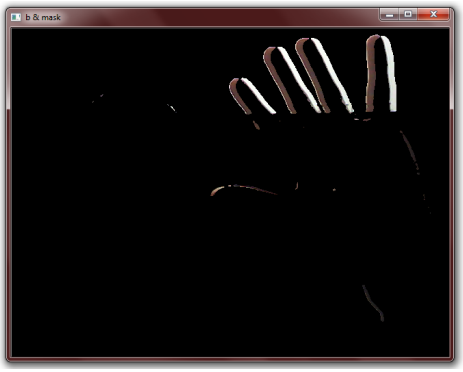
Fatte queste osservazioni, si può comprendere che la nostra figura in movimento nella frame b si può evidenziare attraverso l'espressione:

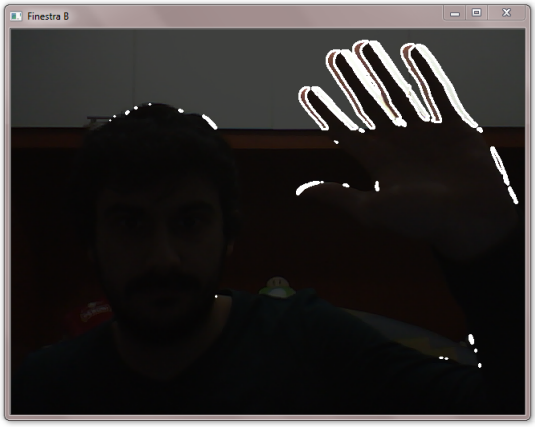
$$\mathbf{b = (b/4 \ \& \ \tilde{}mask) + (b \ \& \ mask)}$$

Infatti, la prima parte del secondo membro è una AND tra b/4 (quindi la frame b oscurata) e ˜mask (che, questa volta, presenterà i contorni e la zona interna in nero, e la parte restante in bianco). Dunque, questa prima parte costruirà una frame contenente la frame b oscurata, con la figura in movimento evidenziata in nero. La seconda parte, invece, costruirà una frame nera in corrispondenza dello sfondo, e invariata nelle zone interne o uguali ai contorni. La loro somma, dunque, sarà proprio la frame b oscurata al di fuori dei contorni, e invariata al loro interno e in loro corrispondenza.

Queste sono le frame coinvolte nell'operazione:







Non resta altro da fare che evidenziare nuovamente i contorni nella frame b attraverso il metodo *drawContours*, per poi restituire vero se la dimensione del vettore intruders è maggiore di zero, falso altrimenti.

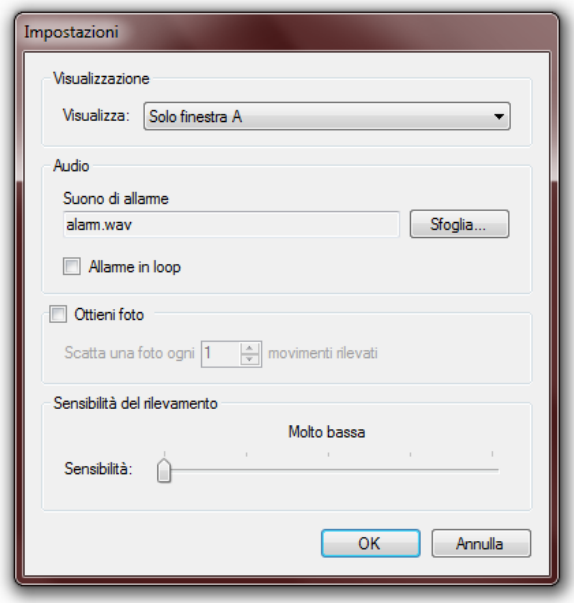
Capitolo 6

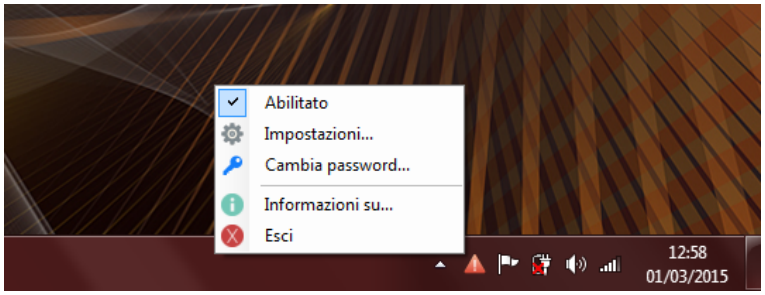
Altre funzionalità

L'applicazione permette di modificare alcune impostazioni per ampliare le sue funzionalità. In particolare, si può:

- nascondere una delle due finestre (A o B) o visualizzarle entrambe;
- selezionare il file da riprodurre come allarme in caso di rilevamento (file WAV);
- riprodurre ciclicamente il suono di allarme a partire dall'istante del rilevamento (o riprodurre il singolo suono ad ogni rilevamento);
- ottenere delle foto degli intrusi con una certa frequenza;
- regolare la sensibilità del rilevamento (in base all'area delle figure individuate dai contorni);
- cambiare la password di identificazione;
- visualizzare informazioni riguardo al progetto e agli sviluppatori.

Nello specifico, sono due le funzionalità che vengono realizzate tramite OpenCV: la possibilità di scattare foto e la regolazione della sensibilità.





6.1 Cattura del rilevamento (takePhoto)

È stata realizzata una funzione a tal scopo. La sua intestazione è:

void *takePhoto*(cv::Mat& **picture**, int **index**)

dove **picture** è la frame da salvare e **index** è il numero identificativo della foto. Prima di poter procedere al salvataggio della foto sul disco, è necessario impostare i parametri relativi al metodo di compressione utilizzato. A tal proposito, si crea un vettore chiamato **compression_params**, nel quale vengono inseriti prima il formato (si è scelto CV_IMWRITE_PNG_COMPRESSION) e poi il rapporto di compressione (9).

Arrivati a questo punto, non resta altro da fare che invocare la funzione *imwrite*:

bool *imwrite*(const String& **filename**, InputArray **img**, const vector<int>& **params**)

dove **filename** è una stringa contenente il nome del file (è stato scelto IMG_index), **img** è la frame da salvare nel file (quindi picture) e **params** è il vettore dei parametri (ossia compression_params).
Dopo aver invocato questa funzione, nella cartella dell'eseguibile, sarà presente quindi il file contenente la frame in questione, in formato PNG.

6.2 Sensibilità di rilevamento

La sensibilità di rilevamento è molto importante nell'algoritmo *intruderAlarm*, in quanto, in base ad essa, vengono rilevati determinati movimenti: con una sensibilità molto bassa, verranno catturati solo movimenti bruschi; aumentandola, invece, verranno percepiti anche movimenti molto lenti.
Alla base di questo parametro vi è principalmente l'area delle figure individuate dai contorni: più la sensibilità sarà alta, minore sarà il valore di tale area per cui la figura viene considerata in movimento. Tale decisione deve essere presa, ovviamente, nell'algoritmo *intruderAlarm*, nel momento in cui tutti i contorni delle figure in movimento sono stati rilevati e inseriti nel vettore **contours**. Il vettore **intruders** conterrà quindi i contorni di tali figure.

L'istruzione in questione è la seguente:

if (area >= ((MAX_SENSIBILITY - *globalSettings.getSensibility*( )) * SENSIBILITY_DIFF))
*intruders.push*_back(**contours[i]**);

dove MAX_SENSIBILITY è il valore massimo della sensibilità (MAX_ SENSIBILITY = 5), *globalSettings.getSensibility*( ) è la funzione che restituisce la sensibilità impostata dall'utente (tramite la finestra Impostazioni) e SENSIBILITY_DIFF è un parametro che indica di quanto la soglia dell'area per il rilevamento varia tra un livello di sensibilità e quello immediatamente successivo (si è posto SENSIBILITY_DIFF = 5000). Dunque, se l'area della figura all'interno dei contorni rilevati è maggiore o uguale a quella soglia, tali contorni vengono inseriti nel vettore **intruders**, e quindi verrà rilevato il movimento.
Da notare che, nel caso di sensibilità massima, la soglia diventa nulla, quindi verrà rilevato anche un minimo movimento.

Capitolo 7

Programma principale

Nel programma principale, sono presenti tutte le istruzioni che concorrono a realizzare il programma. Per prima cosa, si tenta di aprire un collegamento con la webcam: nel caso in cui questa operazione fallisse, il programma terminerebbe. In caso contrario, si entrerebbe in un ciclo infinito, che terminerebbe solo uscendo dal programma attraverso l'apposita voce presente nel menù contestuale.
Se il programma non è attivato, si ha un ciclo vuoto. Altrimenti, vengono prese le due frame per il confronto: se si tratta del primo ciclo, queste vengono prese entrambe dalla webcam, altrimenti viene presa solo la seconda, mantenendo l'ultima frame immediatamente precedente.
A questo punto, viene chiamata la funzione *intruderAlarm* su queste due frame. Se viene rilevato un oggetto in movimento, allora viene fatto suonare l'allarme, secondo le modalità specificate; eventualmente, vengono anche scattate delle foto con la frequenza scelta dall'utente.
Ovviamente, verranno visualizzate solo le finestre impostate dall'utente.