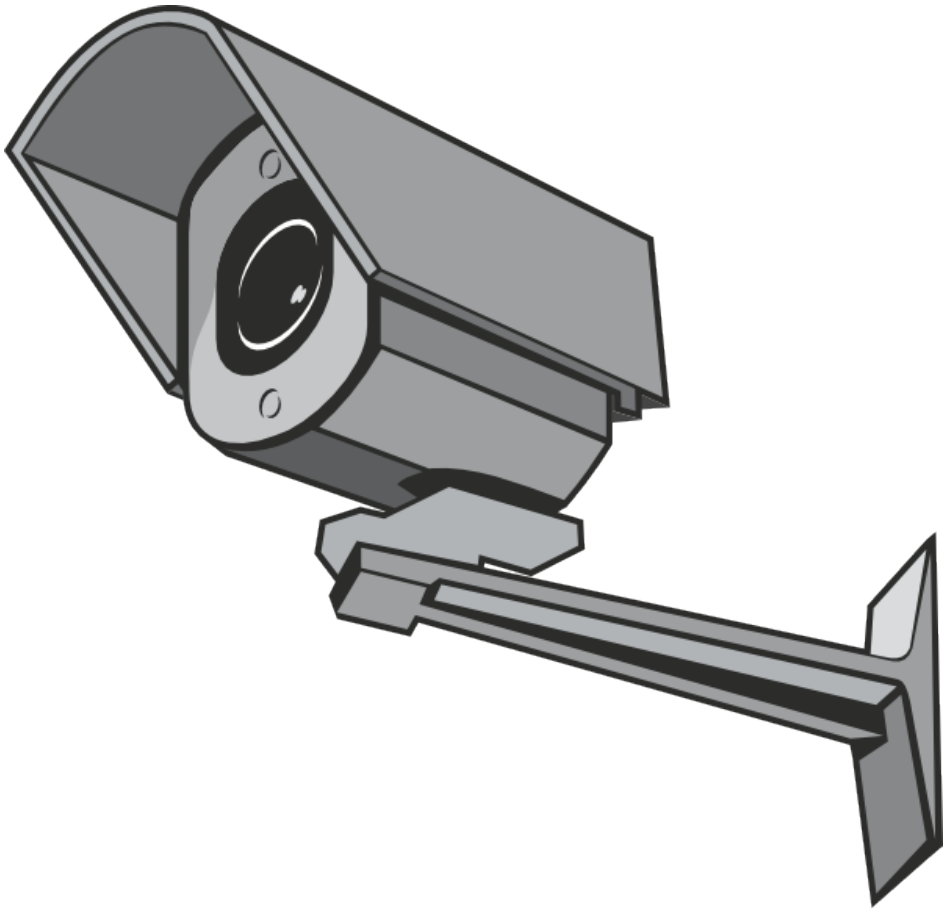
Videosorveglianza movement triggered

Capuano Andrea, De Prisco Mattia, Esposito Gabriella

21 Luglio 2014



Ringraziamenti

Si ringrazia il Prof. Antonio Picariello per la disponibilità e l'aiuto fornitoci durante lo sviluppo e la realizzazione di questo progetto (ringraziamo anche il povero tesista che ci ha fatto da cavia in una delle fasi di test con il prof.)

Contents

1 Introduzione:

1.1 Obiettivi

L'obiettivo di questo progetto è quello di creare un sistema di videosorveglianza che, attraverso una webcam come sensor layer, riconosca automaticamente ed in tempo reale il movimento di oggetti, persone o animali sulla scena. Esso infatti è strutturato in maniera tale da catturare le immagini e salvarle in un file GIF solo quando il sensore rileva un movimento significativo. Il sistema prevede infatti che ci sia un sensore fisso puntato sulla zona d'interesse in modo da poter distinguere ciò che si muove da ciò che è fisso sullo sfondo.

1.2 Possibile utilizzo

Questo sistema di videosorveglianza, per le sue caratteristiche funzionali, è progettato per essere utilizzato in ambienti chiusi tipo casa, ufficio ecc. In particolare potrebbe essere utilizzato per sorvegliare un animale domestico (ad es. un cane lasciato solo in casa) o per sorvegliare una stanza di un appartamento/ufficio. Esso infatti, per come è strutturato, effettua una vera e propria "selezione" di immagini che possono essere d'interesse registrando appunto solo in presenza di movimento

1.3 Possibili problematiche

La prima problematica riscontrata è stata quella di utilizzare un sottrattore di sfondo dinamico, in quanto gli strumenti di registrazione utilizzati (webcam) sono altamente inaffidabili: in particolare sono eccessivamente sensibili alle variazioni di luce e hanno sensibilità variabile a seconda della scena registrata. In una prima versione del programma il movimento della base su cui era poggiata la webcam causava una registrazione ininterrotta in quanto lo sfondo catturato all'inizio veniva modificato. Per risolvere tale problematica è stato utilizzata una combinazione particolare dei valori del backgroundSubtractor (si veda lo studio della classe Sorverglianza.java nei successivi paragrafi) , insieme ad un appropriato valore di learningRate , ovvero la velocità con la quale il programma "impara" il nuovo sfondo.

La seconda problematica è stata quella della scelta del formato della registrazione dei video, in particolare il programma è stato pro-

gettato sin dall'inizio per essere utilizzato con hardware economico al fine di impiegarlo in soluzioni "fai da te" (ad esempio la sorveglianza di un animale domestico). Quindi bisognava tener conto del problema delle dimensioni dei video generati. Un primo approccio all'affrontare la problematica è quello di utilizzare una compressione video. Tuttavia tale soluzione presentava sia un problema di complessità computazionale e conseguente rallentamento in fase di registrazione, sia il problema che c'erano diversi video di lunghezza molto breve che risultava in una tediosa apertura di software per la riproduzione. Dunque la scelta di progetto effettuata è quella di utilizzare il noto formato GIF che supporta le animazioni. Poichè un video non è altro che una sequenza di immagini si utilizza un thread apposito per il salvataggio di queste ultime e il loro trasferimento su una libreria di conversione di un array di immagini in una unica GIF animata.

La terza problematica è stata quella della scelta del numero di punti per rilevare un movimento significativo. Inizialmente si è pensato di effettuare una calibrazione della webcam basata sui valori di picco. Tuttavia questa soluzione si è rivelata presto inefficiente a causa dell'imprecisione del registratore utilizzato (la webcam) che è soggetto ad alta inaffidabilità. Di conseguenza si ottenevano calibrazioni che non rispecchiavano l'effettiva quantità di movimento. Quindi è stata introdotta la possibilità di scegliere una percentuale di punti significativi per il movimento, modificabile manualmente dall'utente attraverso uno slider, che considera come percentuale di riferimento la larghezza della risoluzione della webcam (ad es. se su un immagine 640x480 volessi considerare il 50% dei punti significativi dovrei calcolare il 50% di 640, ovvero 320).

2 Sistema di videosorveglianza

2.1 Struttura e caratteristiche del sistema

Il sistema è strutturato in maniera da avere a livello sensor layer una telecamera o webcam che cattura le immagini. Essa si occupa infatti della raccolta di informazioni dell'ambiente osservato e fornisce un flusso di dati proporzionale alla qualità e alla quantità di immagini catturate. La qualità è dovuta alla tecnologia del sensore mentre la quantità di immagini catturate viene regolata dall'algoritmo di registrazione implementato. Infatti solitamente i flussi di dati di tipo video sono di grosse dimensioni, per ridurre la quantità di frame che non sono d'interesse e utilizzare meno memoria possibile è stato implementato un algoritmo "selettivo" che riduce la ratio di acquisizione. Questo algoritmo infatti prevede che il sistema registri solo le immagini in cui è stato rilevato un movimento significativo, dove per movimento significatico intendiamo una variazione in percentuale di punti di un corpo rispetto a quelli rilevati sullo sfondo.

Il livello image processing layer invece si occupa della raccolta dei dati grazie al modulo image processing system (IPS). Esso a livello object segmentation, infatti, è composto da un algoritmo di backgroud subtractor con refreshing dinamico dello sfondo mentre a livello di tracking implementa un algoritmo non basato su modello geometrico.

3 Strumenti utili alla realizzazione del progetto

La realizzazione dell'elaborato è stata articolata e ha comportato l'utilizzo di diversi strumenti software.

3.1 Ambiente di sviluppo

Il software utilizzato per implementare il progetto è l'ambiente IDE Eclipse Juno, sviluppato per il linguaggio Java. I test di funzionamento sono stati svolti su diversi calcolatori utilizzando diversi tipi di videocamere (built in e non). L'applicazione è stata testata sia in ambienti aperti che chiusi con diverse esposizioni alla luce, il risultato di tali test ha reso evidente che tale applicazione risulta più efficiente al chiuso che all'aperto.

3.2 Componenti hardware

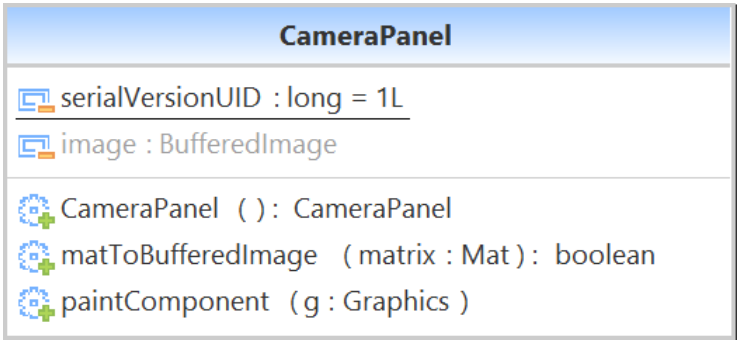
Per la realizzazione di questo progetto, nato come un sistema di videosorveglianza "fai da te", è stato sufficiente utilizzare una webcam collegata ad un calcolatore. Per questo motivo l'intera implementazione è stata ottimizzata per un sistema che prevede componenti hardware a basso costo.

4 Implementazione del progetto

Di seguito verrà riportata l'implementazione di tutto il progetto andando ad analizzare tutti quelli che sono gli elementi fondamentali dell'elaborato.

4.1 Le Classi

4.1.1 La classe CameraPanel



La classe cameraPanel è responsabile della conversione di una matrice di pixels in un'immagine. In particolare, una matrice di pixels è rappresentata attraverso la classe "Mat" mentre un'immagine può essere incapsulata nel tipo "BufferedImage". Il cuore del codice è racchiuso in due istruzioni:

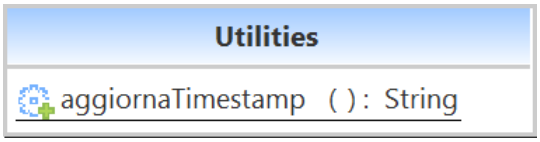
```
Highgui.imencode(".jpg", matrix, mb);
this.image = ImageIO.read(new ByteArrayInputStream(mb.
    toArray()));
```

La prima istruzione specifica il formato di salvataggio, la matrice di origine (*matrix*) e quella di destinazione (*mb*) affinchè possa essere spalmato su un buffer di memoria il contenuto di *matrix*. La seconda istruzione utilizza uno stream di byte per ricevere in input la matrice trasformata in array di byte, dunque attraverso il metodo *read* della metodo statica *ImageIO* acquisiamo l'immagine. Dove *this.image* è una variabile di classe di tipo *BufferedImage* e il metodo *toArray* realizza l'effettiva conversione

```
MatOfByte->byte[];
```

Infine il metodo *paintComponent* è utilizzato per visualizzare l'immagine sull'applicazione. Poichè l'intera classe eredita da **JPanel** possiamo ridefinire tale metodo per eseguire una *drawImage* sull'immagine precedentemente salvata nel *BufferedImage*.

4.1.2 La classe Utilities



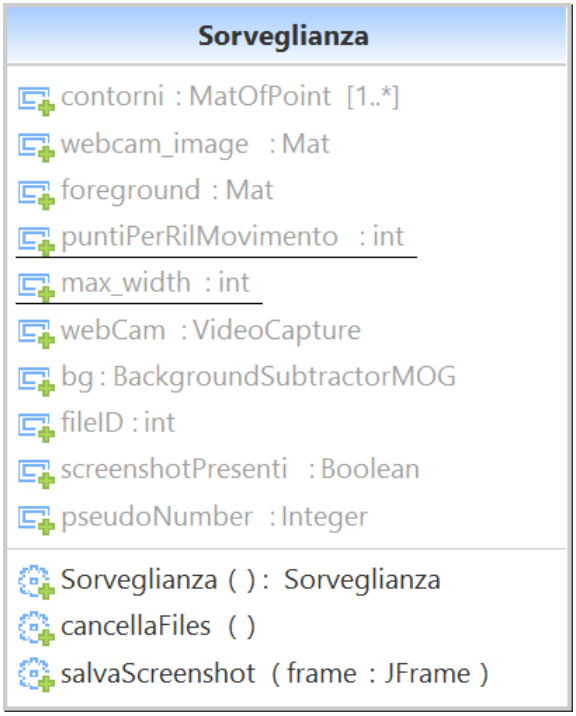
La classe utilities fornisce un metodo di utilità (*aggiornaTimestamp();*) utilizzato da piu' classi. E' sicuramente importante ,al fine di ottenere un sistema di videosorveglianza affidabile, riuscire a capire l'istante in cui ha inizio la registrazione video. Per tale motivo questa funzione viene utilizzata sia in fase di acquisizione video, dove sarà presente secondo per secondo l'orario preciso di registrazione, sia in fase di salvataggio del video; infatti il File che verrà creato sarà nominato in base alla data e l'ora di registrazione.

Il metodo *aggiornaTimestamp();* restituisce una Stringa contente l'orario nel seguente formato :

"GIORNO-MESE-ANNO ORA.MINUTI.SECONDI"

ed utilizza la classe Java **Calendar** per l'ottenimento dell'orario e la classe **SimpleDataFormat** affinchè l'input possa essere formattato in maniera desiderata. In particolare viene utilizzato il metodo *cal.getTime()* (dove *cal* è un'istanza di Calendar) per ottenere l'orario corrente.

4.1.3 La classe Sorveglianza



Appena istanziata, la classe **Sorveglianza**, si occupa di caricare la libreria di **openCV**:

```
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
```

Successivamente istanzia una serie di variabili che avranno lo scopo di regolare l'acquisizione e il salvataggio degli screenshot. Tra queste riveste molta importanza *puntiPerIlMovimento* modificabile tramite lo slider durante l'esecuzione del programma. Tale variabile permette di variare la soglia di cattura delle immagini, definendo quanto ampio debba essere il movimento affinchè esso venga ritenuto significativo. Viene poi effettuato il collegamento al sensore e viene inizializzato il *BackgroundSubtractor* tramite le istruzioni:

```
this.webCam = new VideoCapture(0);
this.bg = new BackgroundSubtractorMOG(200, 5, 0.7, 0);
```

Il costruttore del BackgroundSubtractor ci permette di definire 4 parametri:

1. history: regola il rate di adattamento dello sfondo(all'aumentare di questo valore diminuirà il rate di adattamento).
2. nmixtures: massimo numero di misture gaussiane permesse.
3. backgroundRatio: soglia che definisce se un componente è abbastanza significativo per essere incluso nel modello di background.
4. noiseSigma: intensità del rumore.

Sorveglianza definisce poi due metodi:

```
public void salvaScreenshot(JFrame frame) throws IOException;
```

che si occupa di salvare in un file jpg il contenuto attuale della *JFrame*, in particolare:

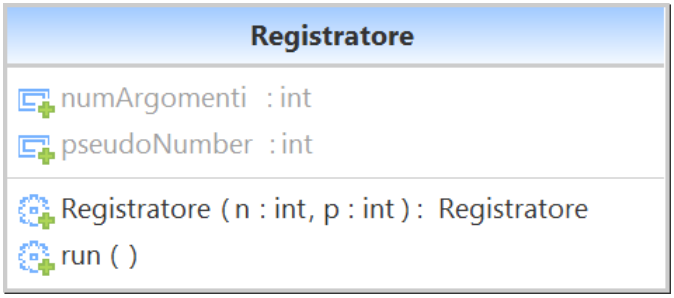
```
BufferedImage image=(BufferedImage)frame.createImage(
    frame.getSize().width, frame.getSize().height);
Graphics g = image.getGraphics();
frame.paint(g);
g.dispose();
ImageIO.write(image, "jpg", new File("MyFrame"+this.
    pseudoNumber+"-" + this.fileID++ + ".jpg"));
this.screenshotPresenti=true;
```

Si crea prima una *BufferedImage* avente le stesse dimensioni della *JFrame*, per poi copiarne il contenuto in essa. Infine viene creato il file jpg il cui nome è generato tramite uno *pseudoNumber*, diverso per ogni file, e viene segnalata al programma principale la presenza di screenshot.

```
public void cancellaFiles();
```

funzione chiamata al termine del programma, che effettua la cancellazione di tutte le immagini jpg salvate durante la sua esecuzione, cercando i nomi di tutti i file terminanti con ".jpg".

4.1.4 La classe Registratore



La classe **Registratore** è un thread che unisce gli screenshot salvati dalla classe "Sorveglianza" in un file gif.

Prima di iniziare viene effettuato un controllo:

```
if(numArgomenti>min_screenshot);
```

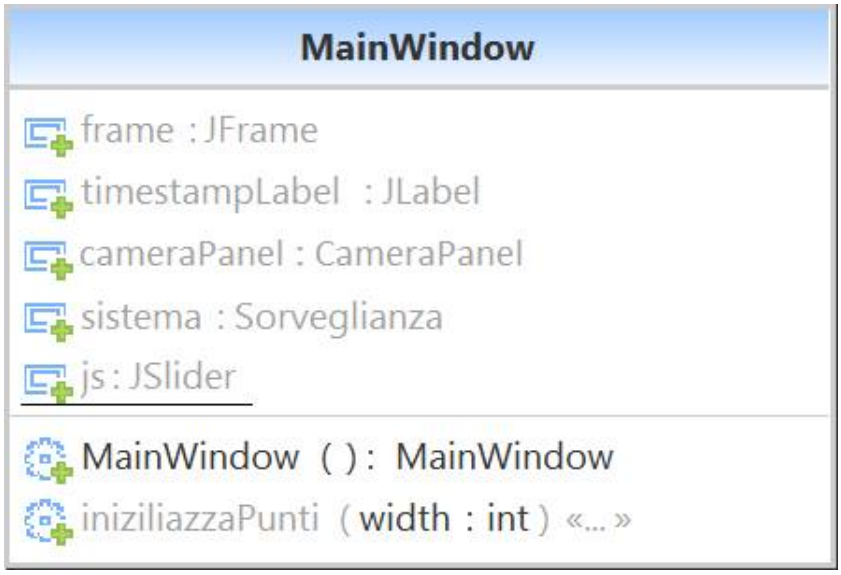
Dove *min_screenshot* è una variabile preimpostata con la quale si verifica che ci sia un numero minimo di screenshot salvati, ovvero che sia stato registrato un movimento significativo. In caso affermativo si procede a generare un vettore contenente i nomi di tutti gli screenshot da unire nel file gif (i nomi vengono ricavati in base a *"numArgomenti"*, il numero di screenshot, e *pseudoNumber*, variabile usata da Sorveglianza per salvare screenshot con nomi diversi).

Tale vettore viene poi passato alla libreria **GifSequenceWriter**:

```
GifSequenceWriter.main(argomenti);
```

la quale si occupa dell'effettiva conversione dei file jpg in una gif.

4.1.5 La classe MainWindow



Classe adibita all'inizializzazione e gestione di tutte le variabili necessarie alla corretta esecuzione e visualizzazione della finestra. In particolare essa è composta da:

1. Un *JFrame* che conterrà tutti gli elementi dell'interfaccia grafica.
2. Un *cameraPanel* sul quale verrà visualizzata la ripresa del sensore.
3. Un *JLabel* contentente il timestamp costantemente aggiornato.
4. Un *JSlider* che regola il valore di *puntiPerIlMovimento* modificandolo in percentuale rispetto alle dimensioni dello sfondo:

```
int percentuale = (Sorveglianza.puntiPerIlMovimento*100)/
    Sorveglianza.max_width;
```

Contiene poi il metodo *public void iniziliazzaPunti(int width)*; , adibito all'inizializzazione dei valori del *JSlider*.

4.1.6 La classe Main

La classe Main è il cuore dell'applicazione e possiede il metodo statico main essenziale all'avvio del programma. Le prime istruzioni sono necessarie all'inizializzazione delle componenti MainWindow (la finestra effettiva) e Sorveglianza che incapsulano i metodi utili al funzionamento del software:

```
MainWindow window = new MainWindow();
Sorveglianza s = window.sistema;
```

Dopodichè si avvia un pool di Thread necessario per la corretta gestione dei Threads che effettueranno il salvataggio del video (si veda riferimento a classe **Registratore**). Prima dell'avvio effettivo del ciclo while di cattura video, è necessario effettuare una check del corretto funzionamento della webcam tramite l'istruzione:

```
window.sistema.webCam.isOpened()
```

Nel caso di lettura errata della webcam l'applicazione viene immediatamente interrotta, altrimenti si attendono 500ms per l'inizializzazione del dispositivo e si entra nel ciclo while che si interromperà solo alla chiusura dell'applicazione.

La fase effettiva di lettura dalla webcam avviene nella seguente istruzione:

```
s.webCam.read(s.webcam_image);
```

dove *s.webCam* è un istanza di una classe **VideoCapture** nativa di OpenCV necessaria per l'apertura di uno stream video dalla webCam. Arrivati a questo punto, dopo una breve fase di inizializzazione dello slider, chiamata attraverso il metodo *window.inizializzaPunti();* ,abbiamo la procedura di acquisizione che consiste in tre passaggi fondamentali:

1. Applicazione del Background subtractor.
2. Ricerca dei punti di movimento rispetto al background.
3. Salvataggio dell'immagine acquisita se questa soddisfa il criterio di movimento significativo.

FASE-1 Applicazione del Background Subtractor.

Ricordando che "s.bg" è un oggetto di tipo **BackgroundSubtractorMOG**, andiamo a descrivere il metodo *apply*:

```
BackgroundSubtractor.apply(Mat image, Mat fgmask, double learningRate);
```

Parametri:

image: L'immagine sucessiva sulla quale si applica il background subtractor.

fgmask: La maschera di output memorizzata con 8-bit.

learningRate: Con quanta velocità il sottrattore di sfondo si adatta ai cambiamenti.

Dunque si ha :

```
s.bg.apply(s.webcam_image, s.foreground, 0.1);
```

Sostituendo al valore di *image* la frame appena catturata dalla webcam , al valore *fgmask* il nostro foreground e una *learning rate* il cui valore è stato scelto a seguito di "trial and error".

FASE-2 Ricerca dei punti di contorno di movimento.

A questo punto è possibile andare a verificare se c'è un'effettiva differenza tra lo sfondo e l'immagine appena catturata dalla webcam. Per fare ciò si utilizza la funzione findContours:

```
void org.opencv.imgproc.Imgproc.findContours(Mat image,
    List<MatOfPoint> contours, Mat hierarchy, int mode,
    int method)
```

La funzione estrae i contorni da un'immagine utilizzando l'algoritmo [Suzuki85]. I contorni sono uno strumento utile per le analisi delle forme, per fare object detection e recognition.

Parametri:

image: Source, L'immagine source a 8 bit.

contours: I contorni trovati (Vengono piazzati in un vettore).

hierarchy: Un vettore di output opzionale contente informazioni sulla topologia dell'immagine.

mode: Come effettuare il retrieval.

method: come approssimare i contorni.

La funzione è stata applicata come segue:

```
Imgproc.findContours(s.foreground, s.contorni, new Mat() , 0, 1);
```

Quindi si è scelto di NON utilizzare un output specifico per la topologia (New Mat() non è salvata in una variabile) e di impostare mode = 0 ovvero il retrieval è indipendente dalla topologia dell'immagine e method = 1 ovvero NON viene effettuata un'approssimazione dei contorni.

FASE-3 Salvataggio dell'immagine acquisita se questa soddisfa il criterio di movimento significativo.

Quest'ultima fase è quella che avvia la funzione *salvaScreenshot* nel caso in cui ci sia effettivo movimento:

```
if(s.contorni.size() > Sorveglianza.puntiPerRilMovimento)
        {
                s.salvaScreenshot(window.frame);
        }
```

s.contorni è il vettore di contorni che è stato riempito nella precedente fase da *findContours*. Il valore di soglia (*puntiPerilMovimento*) è scelto inizialmente in maniera automatica dal programma, in base alle dimensioni della frame catturata, e poi è modificabile manualmente attraverso uno slider.

Nel caso in cui non ci sia piu' movimento significativo viene effettuato un controllo sul numero di screenshots scattati. Se sono presenti viene avviato il thread che si occupa della fase di registrazione:

```
if(s.screenshotPresenti==true)
        {
        pool.execute(new Registratore(s.fileID, s.pseudoNumber));
        }
```

Alla chiusura viene eseguita una *webcam.release()* per rilasciare la webcam e si aspetta la terminazione dei threads.

Infine viene chiamata la funzione di eliminazione delle singole immagini (jpg).

4.2 Diagramma UML