Corso di Laurea Magistrale in Informatica

Relazione sul progetto di Multimedia

CVideoPlayer: feature FAST, SIFT e SURF.

Professore: Sebastiano Battiato
Autore: Daniele Licitra, M13 000090

Introduzione

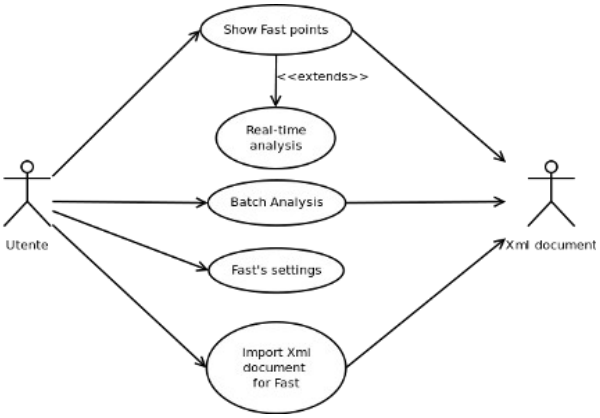
Il progetto implementa i metodi necessari all'utilizzo delle feature FAST, SIFT e SURF all'interno del software CVideoPlayer.
Il metodo *FAST Corner Detection* si occupa di ricercare angoli all'interno delle immagini in scala di grigi. Di solito è un passo preliminare ad elaborazioni di Computer Vision. La ricerca si svolge candidando ad angolo un punto *"pivot"* e valutando un suo intorno di 16 pixel. Si verificano i valori dei pixel dell'intorno e se un'alta percentuale di pixel consecutivi (almeno 12) ha un valore che differisce almeno di una soglia *t* dal valore del pixel pivot, allora siamo in presenza di un angolo.
Il metodo *Scale Invariant Feature Trasform (SIFT)* calcola un insieme di vettori caratteristici di un'immagine in scala di grigi. I vettori forniti sono invarianti a traslazioni, scaling e rotazioni dell'immagine; questo è ottenuto in parte applicando una *DoG, difference of Gaussian*. L'approccio consiste nell'identificazione di punti chiave (*keypoint*) come massimi e minimi della funzione DoG. Successivamente, per ogni keypoint vengono calcolati i vettori che descrivono la regione locale del keypoint; questi vettori vengono chiamati *descrittori*. Date due immagini, se almeno 3 keypoint coincidono con un approccio *Nearest-Neighbour*, allora c'è un'alta probabilità che l'oggetto descritto dai keypoint sia il medesimo.
Il metodo *Speeded Up Robust Feature (SURF)* calcola i vettori caratteristici di un'immagine in scala di grigi più velocemente del metodo SIFT e con maggior resistenza al rumore.
Il software fornisce i metodi per il calcolo real-time delle features, la creazione di un file XML dall'analisi in modalità batch, l'importazione di un file XML e le finestre per impostare i valori che regolano il funzionamento delle features.

I casi d'uso

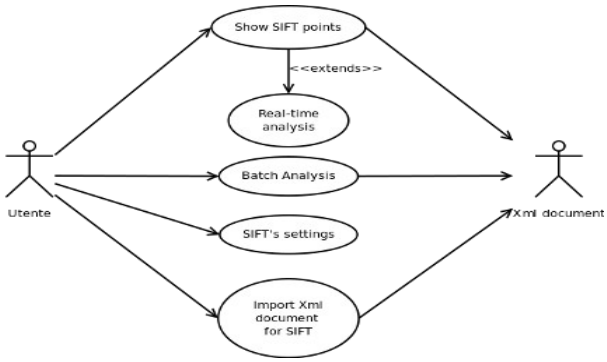
I casi d'uso sono simili per le tre feature:

- **Visualizzazione punti a video:** l'utente mette la spunta nella casella della relativa feature a riproduzione in corso o avvia la riproduzione in seguito. Nel caso in cui non sia stato ne calcolato ne importato un file xml, viene eseguita un'analisi real-time dei frame, altrimenti le informazioni vengono importate dal documento xml;
- **Analisi batch:** l'utente seleziona dal menù a tendina la voce "*Analyze Video*". Verrà avviata l'analisi frame per frame del video, la scrittura delle caratteristiche rilevate su file XML e verrà mostrata a schermo una finestra contenente la barra del progresso. Al termine verranno salvati i riferimenti ai frame in cui sono state rilevate feature;
- **Importazione file XML:** l'utente può scegliere di importare un file XML precedentemente calcolato per la feature in questione mediante l'apposita voce nel menù a tendina "*Import XML file*";
- **Impostazione parametri delle feature:** dall'apposito menù a tendina, l'utente seleziona la voce *"Setting"*e viene visualizzata la finestra per l'impostazione dei parametri. Premendo *Ok* l'utente conferma i nuovi valori altrimenti le modifiche verranno ignorate;

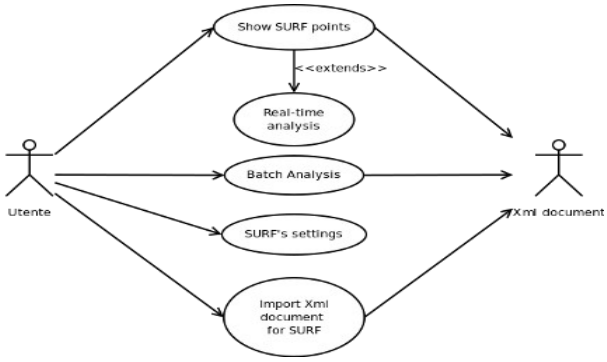
Di seguito riporto il diagramma dei casi d'uso per FAST:



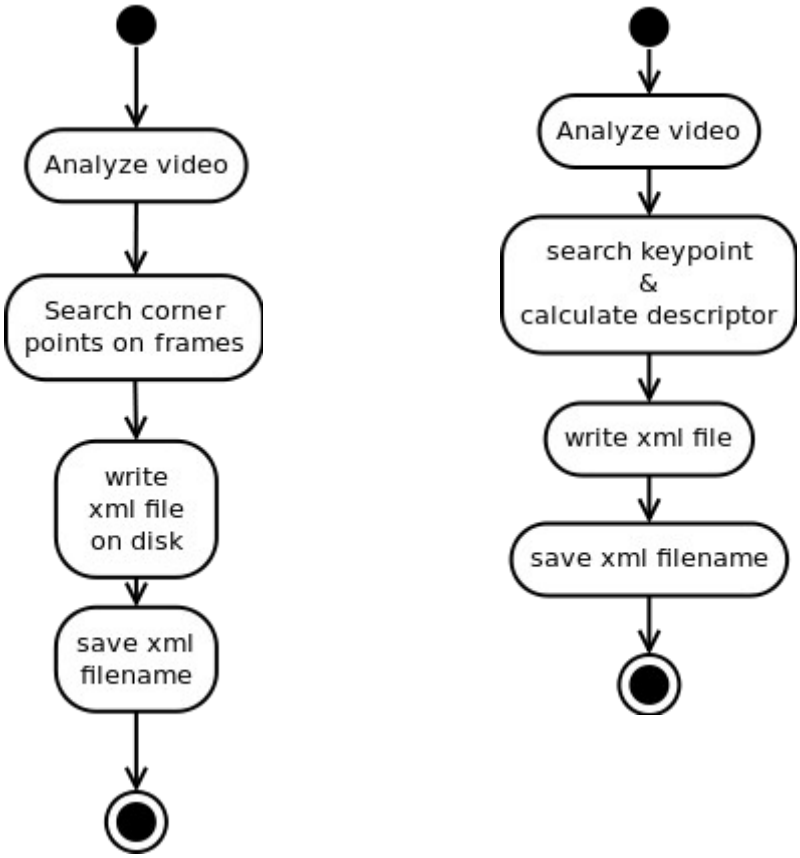
Riporto il diagramma dei casi d'uso per SIFT:



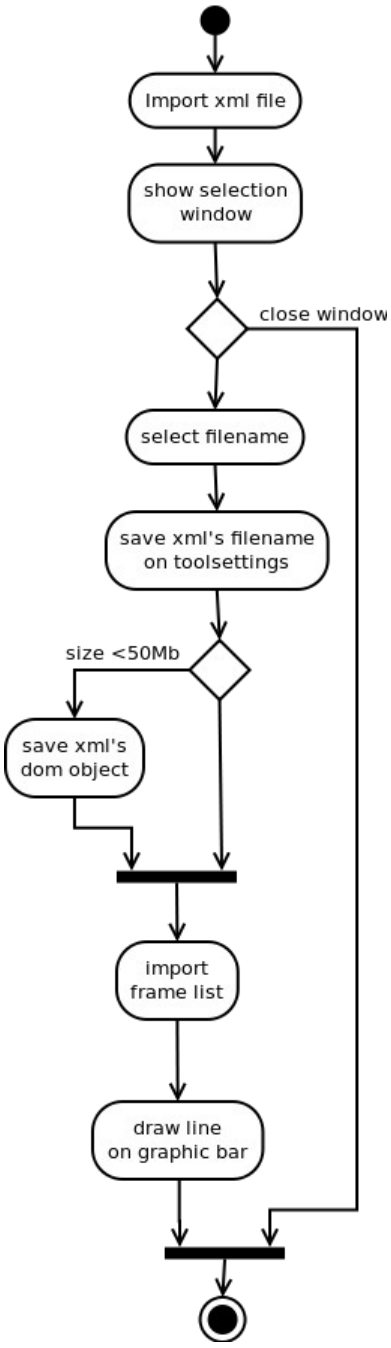
Qui di seguito il diagramma dei casi d'uso per SURF:

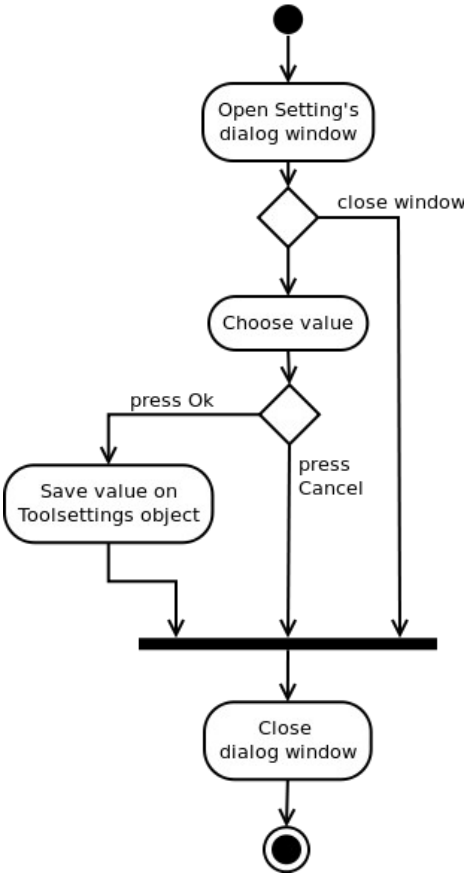


Nel caso di analisi, il primo diagramma rappresenta l'analisi mediante l'algoritmo FAST, mentre il secondo diagramma rappresenta l'analisi con i metodi SIFT e SURF. Il procedimento è simile per le tre features, le seconde hanno in più il calcolo dei descrittori.

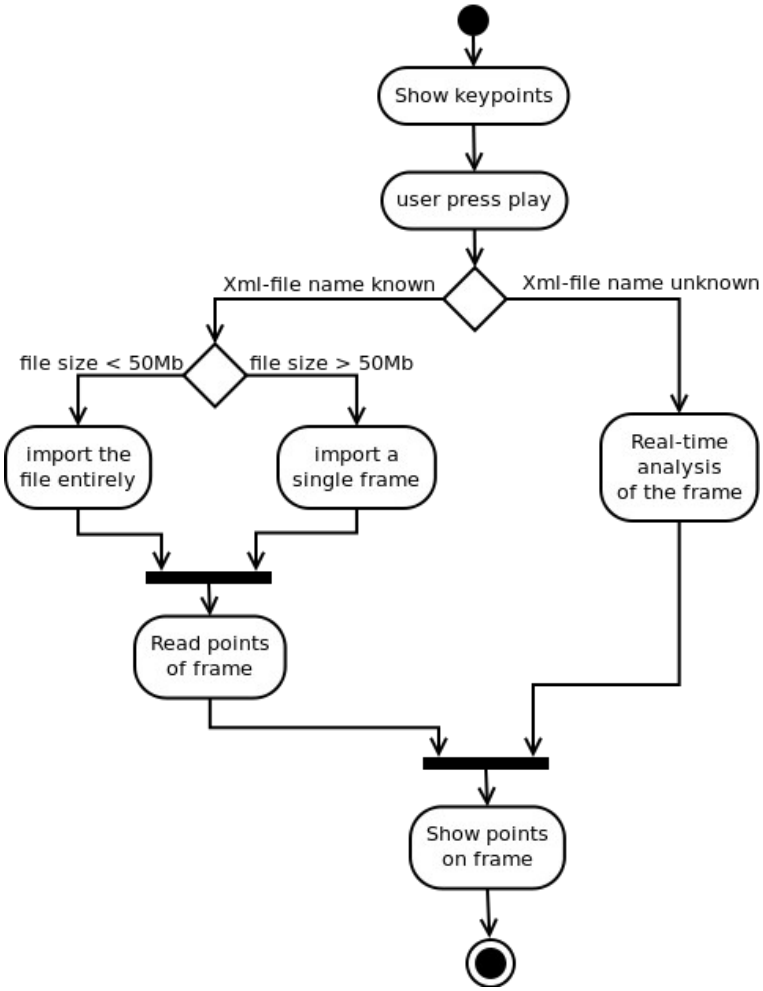


Sia nel caso di importazione del file XML (diagramma di sinistra) che di impostazione dei parametri delle feature(diagramma di destra), i passi da eseguire sono i medesimi per le tre caratteristiche.
Da notare nell'importazione il diverso flusso nel caso di file più grandi o più piccoli di 50 Mb. Il DomDocument viene importato per intero solamente se il file è di piccole dimensioni, altrimenti vieni importato il singolo frame corrente. Nella fase di importazione, che può durare alcuni secondi, viene disabilitata la finestra del programma in modo da informare l'utente che l'operazione è in corso e non è ancora stata terminata.





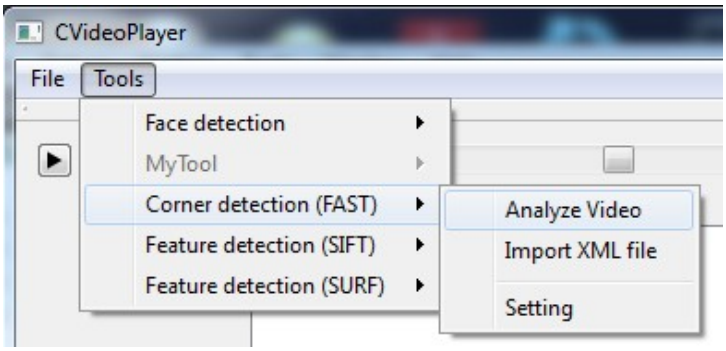
Il caso d'uso di visualizzazione dei keypoint, siano essi generati con FAST, SIFT o SURF, necessita una descrizione più approfondita. L'utente, dopo aver messo la spunta nella casella relativa alla caratteristica da visualizzare, preme il pulsante *play* avviando la riproduzione.
Nel caso in cui il nome del file XML è sconosciuto, per ogni frame è avviata l'analisi in tempo reale.
Nel caso in cui il nome del file XML è noto (ciò è vero se è stato importato un file XML o se è stata appena terminata un'analisi), si verifica la dimensione del file; se il file ha dimensione minore di 50 Mb, viene importato l'intero documento in memoria; se il file ha dimensioni maggiori del limite sopracitato, vengono importate le informazioni relative al solo frame da visualizzare (operazione effettuata ad ogni frame).
Avendo a disposizione adesso le informazioni sui punti da visualizzare nel frame, effettua la visualizzazione a video.



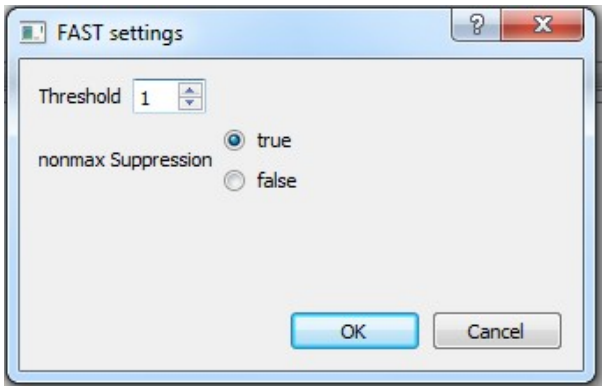
L'interfaccia

Tutte le funzionalità possono essere comodamente raggiunte dal menù a tendina "*Tools"* selezionando la voce relativa alla feature con la quale si vuole operare. I sotto-menù delle voci "*Corner detection (FAST)"*, *"Feature detection (SIFT)"* e "*Feature detection (SURF)"* sono identici e contengono le voci:

- *Analyze Video*: avvia l'analisi in modalità batch per la scrittura del file XML;
- *Import XML file*: mostra la finestra per la selezione del file XML da importare e ne carica i contenuti;
- *Setting*: mostra la finestra per impostare i parametri della feature alla quale fa riferimento.

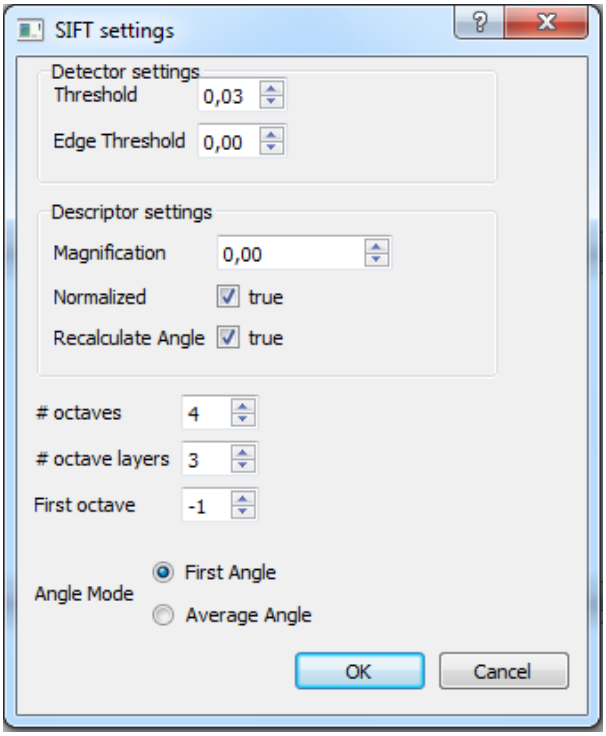


Di seguito riporto le finestre per l'impostazione dei parametri per le tre feature:



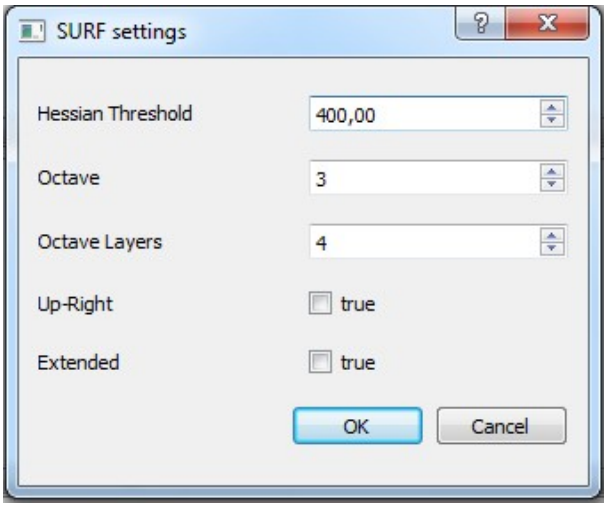
Per la feature FAST possono essere impostati i parametri:

- *Threshold*: valore intero usato come soglia *t* per la differenza tra il valore del pixel centrale e i valori dei pixel dell'intorno da superare affinché si riscontri un angolo (il valore di default è 1, buoni risultati si ottengono per soglie intorno a 15, ma varia molto dal video);
- *nonmax Suppression*: su ogni angolo riscontrato, viene calcolata una "score function" V. Se si imposta *true*, nel caso di angoli adiacenti, si mantiene solo quello con il valore maggiore di V, altrimenti vengono presi tutti gli angoli. Il valore di default è *true*.



Per la feature SIFT, i parametri possono essere raggruppati come parametri che influenzano solo la ricerca, come parametri che influenzano solo il calcolo dei descrittori e come parametri che influenzano entrambi.
I parametri sono :

- *Threshold* : soglia con valore di default 0,03;
- *Edge Threshold* : soglia per l'esclusione di keypoint situati sui confini delle regioni (usata come differenza tra l'autovalore di modulo massimo e l'autovalore di modulo minimo);
- *Magnification :* costante moltiplicativa per la quale vengono moltiplicati i pesi dei valori dell'istogramma*;*
- *Normalized* : se *true,* normalizza i valori dei descrittori;
- *#octaves* : numero di ottave;
- *#octave layers* : numero di livelli per le ottave;
- *First octave* : prima ottava;



I parametri per la feature SURF sono:

- *Hessian Threshold* : soglia da applicare all'hessiano della DoG (default è 400);
- *Octave* : numero di ottave (default è 3);
- *Octave Layers* : numero di livelli per le ottave (default è 4);
- *Extended* : se spuntato, permette di estendere i descrittori per migliorare i dati per regioni invarianti (default è falso);

Sintassi dei file XML

Per ogni feature viene creato un file XML con una specifica sintassi.

I file XML riguardanti la FAST corner detection hanno la seguente sintassi:

`<!DOCTYPE fast_detection>` è il document type che contraddistingue questi file;
`<fast_detection></fast_detection>` è il tag di root ;
`<frame_#></frame_#>` è il tag del frame numero # ;
`<fast_corner x="110" y="3" size="6" octave="0" response="16" class_id="-1" angle="-1"/>` è il tag di un corner trovato con FAST. Contiene le informazioni riguardo l'ascissa e l'ordinata del punto all'interno del frame, il raggio dell'intorno e la "*response*". Gli altri valori sono trascurabili. Tutti i valori sono interi.

Ecco un esempio di documento XML per FAST:

```
<!DOCTYPE fast_detection>
<fast_detection>
  <frame_0>
    <fast_corner x="110" y="3" size="6" octave="0" response="16" class_id="-1" angle="-1"/>
    <fast_corner x="95" y="4" size="6" octave="0" response="17" class_id="-1" angle="-1"/>
    <fast_corner x="8" y="7" size="6" octave="0" response="19" class_id="-1" angle="-1"/>
  ...
  </frame_88>
</fast_detection>
```

I file XML riguardanti la SIFT feature detection hanno la seguente sintassi:

`<!DOCTYPE sift_feature_detection>` è il document type che contraddistingue questi file;
`<sift_feature_detection></sift_feature_detection>` è il tag di root per SIFT;
`<frame_#></frame_#>` è il tag del frame numero #;
`<sift_keypoint x="96.1574" y="44.4003" size="11.2787" octave="0" response="0" sift_descriptors="5.49756e+012 4.19430e+006 ... 0 " class_id="0" angle="-40.6744"/>` è il tag rappresentante il keypoint rilevato con SIFT. Contiene le informazioni spaziali (ascissa, ordinata, angolazione e dimensione) e le informazioni qualitative, ovvero i descrittori. Tutti i valori sono float.

Ecco uno stralcio di documento XML per SIFT:

```
<!DOCTYPE sift_feature_detection>
<sift_feature_detection>
  <frame_0>
      <sift_keypoint   x="96.1574"   y="44.4003"   size="11.2787"   octave="0"   response="0"
sift_descriptors="5.49756e+012 4.19430e+006 ... 0 " class_id="0" angle="-40.6744"/>
  </frame_0>
  <frame_11>
    <sift_keypoint    x="77.3064"    y="43.9271"    size="37.7365"    octave="1"    response="0"
sift_descriptors="7.03688e+013 512 3.22123e+009 ... 131072 " class_id="0" angle="-30.7317"/>
...
  </frame_2054>
</sift_feature_detection>
```

I file XML riguardanti la SURF feature detection hanno la seguente sintassi:

`<!DOCTYPE surf_feature_detection>` è il document type che contraddistingue questi file;
`<surf_feature_detection></surf_feature_detection>` è il tag di root per SURF;
`<frame_#></frame_#>` è il tag del frame numero #;
`<surf_keypoint x="95.6487" y="16.5573" octave="0" size="15" surf_descriptors="-2.18678e-038 8.02747e-026 ... 5.55632e-023 " response="446.092" class_id="-1" angle="10.396"/>` è il tag rappresentante il keypoint rilevato con SURF. Contiene le informazioni spaziali (ascissa, ordinata, angolazione e dimensione) e le informazioni qualitative, ovvero i descrittori e la response. Tutti i valori sono float.

Ecco una parte di un documento XML per SURF:

```
<!DOCTYPE surf_feature_detection>
<surf_feature_detection>
  <frame_0>
  <surf_keypoint x="95.6487" y="16.5573" octave="0" size="15" surf_descriptors="-2.18678e-038
8.02747e-026 ... 5.55632e-023" response="446.092" class_id="-1" angle="10.396"/>
  ...
 </frame_2054>
</surf_feature_detection>
```

La libreria utilizzata, OpenCV

Per implementare le funzionalità è stata usata la libreria OpenCV, libreria opensource che fornisce molti strumenti di Computer Vision.

La feature FAST si occupa di cercare possibili angoli all'interno di una immagine mediante l'algoritmo di Edward Rosten presentato nell'articolo "*Machine learning for high-speed corner detection*". OpenCV ci mette a disposizione la funzione "*FAST*" definita nel modo seguente:

```
void FAST(const Mat& image, vector<KeyPoint>& keypoints, int threshold,
bool nonmaxSupression=true)
```

`Mat& image`: riferimento alla matrice contenente l'immagine in scala di grigi da sottoporre all'algoritmo;
`vector<KeyPoint>& keypoints`: riferimento al vettore di Keypoint nel quale verranno salvati i keypoints(corner) trovati nell'immagine;
`int threshold`: valore intero che indica la soglia per la discriminazione della differenza tra il valore del pixel centrale e i valori dei pixel dell'intorno;
`bool nonmaxSupression`: su ogni angolo riscontrato, viene calcolata una "score function" V. Se si imposta *TRUE*, nel caso di angoli adiacenti, si mantiene solo quello con il valore maggiore di V. Il valore di default è *TRUE*.

Le features SIFT e SURF si occupano in modo differente di descrivere un'immagine selezionando alcuni punti notevoli. Su questi punti vengono calcolati i descrittori. OpenCV ci mette a disposizione quattro oggetti, due per la ricerca dei keypoint e due per il calcolo dei descrittori.

Per quanto riguarda SIFT, i costruttori dell'oggetto detector e descriptor sono rispettivamente (gli argomenti dei costruttori sono stati descritti nella sezione riguardante le interfacce per l'immissione dei parametri):

```
SiftFeatureDetector( double _threshold, double _edgeThreshold,
  int _nOctaves = CommonParams::DEFAULT_NOCTAVES,
  int _nOctaveLayers = CommonParams::DEFAULT_NOCTAVE_LAYERS,
  int _firstOctave = CommonParams::DEFAULT_FIRST_OCTAVE,
  int _angleMode = CommonParams::FIRST_ANGLE );

SiftDescriptorExtractor( double _magnification, bool _isNormalize=true,
  bool _recalculateAngles = true,
  int _nOctaves=CommonParams::DEFAULT_NOCTAVES,
  int _nOctaveLayers=CommonParams::DEFAULT_NOCTAVE_LAYERS,
  int _firstOctave=CommonParams::DEFAULT_FIRST_OCTAVE,
  int _angleMode=CommonParams::FIRST_ANGLE );
```

I metodi da utilizzare per la ricerca dei keypoint e il calcolo dei descrittori sono:

```
SiftFeatureDetector.detect(const Mat& img, vector<KeyPoint>& keypoints);
SiftDescriptorExtractor.compute(const Mat& img, vector<KeyPoint>&
  keypoints, Mat& descriptors);
```

`const Mat& img`: frame del quale calcolare i keypoint o dal quale estrarre i descrittori;
`vector<KeyPoint>& keypoints`: vettore in cui sono salvati i keypoint trovati o da dove vengono letti;
`Mat& descriptors`: matrice che ha per righe i descrittori dei keypoint;

Nell'implementazione, gli argomenti dei costruttori sono presi dalla classe *ToolSettings* nel seguente modo:

```
cv::SiftFeatureDetector detector(settings->get_sift_threshold(),\
     settings->get_sift_edgeThreshold(),\
     settings->get_sift_nOctaves(),\
     settings->get_sift_nOctaveLayers(),\
     settings->get_sift_firstOctave(),\
     settings->get_sift_angleMode());

cv::SiftDescriptorExtractor descriptor(settings->get_sift_magnification(),\
     settings->get_sift_isNormalized(),\
     settings->get_sift_recalculateAngle(),\
     settings->get_sift_nOctaves(),\
     settings->get_sift_nOctaveLayers(),\
     settings->get_sift_firstOctave(),\
     settings->get_sift_angleMode());
```

Per quanto riguarda SURF, i costruttori dell'oggetto detector e descriptor sono rispettivamente (gli argomenti dei costruttori sono stati descritti nella sezione riguardante le interfacce per l'immissione dei parametri):

```
SurfFeatureDetector(double hessianThreshold, int nOctaves=4,
  int nOctaveLayers=2, bool upright=false);
SurfDescriptorExtractor(int nOctaves=4, int nOctaveLayers=2,
  bool extended=false, bool upright=false);
```

I metodi da utilizzare per la ricerca dei keypoint e il calcolo dei descrittori sono:

```
SurfFeatureDetector.detect(const Mat& img, vector<KeyPoint>& keypoints);
SurfDescriptorExtractor.compute(const Mat& img, vector<KeyPoint>&
  keypoints, Mat& descriptors);
```

`const Mat& img`: frame del quale calcolare i keypoint o dal quale estrarre i descrittori;
`vector<KeyPoint>& keypoints`: vettore in cui sono salvati i keypoint trovati o da dove vengono letti;
`Mat& descriptors`: matrice che ha per righe i descrittori dei keypoint;

Nell'implementazione, gli argomenti dei costruttori sono presi dalla classe *ToolSettings* nel seguente modo:

```
cv::SurfFeatureDetector detector(settings->get_surf_hessian_threshold(),\
    settings->get_surf_nOctaves(),\
    settings->get_surf_nOctaveLayers(),\
    settings->get_surf_upright());
cv::SurfDescriptorExtractor descriptor(settings->get_surf_nOctaves(),\
    settings->get_surf_nOctaveLayers(),\
    settings->get_surf_extended(),
    settings->get_surf_upright());
```

Definizione dei metodi

Sono stati aggiunti molti metodi alle classi pre-esistenti di CVideoPlayer. Di seguito elenco i metodi per classe.

La classe *Controller* si occupa dei controlli e di lanciare i metodi della classe *VideoTools*. Elenco brevemente i metodi e i membri della classe:

```
/* Metodi FAST */
void analyzeFAST();
void showFast(bool b);
/* Metodi SIFT */
void analyzeSIFT();
void showSift(bool b);
/* Metodi SURF */
void analyzeSURF();
void showSurf(bool b);
bool fast;
bool sift;
bool surf;
```

La classe *MainWindow* si occupa della gestione dell'interfaccia grafica. Contiene i metodi che gestiscono gli eventi scatenati dall'interazione con l'interfaccia. Elenco brevemente i metodi e le istanze delle classi utilizzate per l'immissione dei parametri

```
/* Metodi FAST */
void on_checkBoxFast_stateChanged(int arg1);
void on_actionAnalyzeFAST_triggered();
void on_actionSettingFAST_triggered();
void on_actionImportFAST_triggered();
/* Metodi SIFT */
void on_actionAnalyzeSIFT_triggered();
void on_actionImportSIFT_triggered();
void on_checkBoxSIFT_stateChanged(int state);
void on_actionSettingsSIFT_triggered();
/* Metodi SURF */
void on_checkBoxSURF_stateChanged(int state);
void on_actionSettingsSURF_triggered();
void on_actionAnalyzeSURF_triggered();
void on_actionImportSURF_triggered();
/* Istanze delle classi per i settings */
settingFAST fast_dialog;
SettingSift sift_dialog;
SettingSurf surf_dialog;
```

La classe *ToolSettings* ha come membri i parametri e le impostazioni per i vari strumenti. I metodi della classe sono tutti per l'impostazione e la lettura dei membri.

```
// FAST
int getFASTthreshold(){return fast_threshold;}
bool getFASTnonmaxSuppression(){return fast_nonmaxSuppression;}
QString getFAST_xmlfilename(){return xml_fast_filename;}
void setFASTthreshold(int value);
void setFASTnonmaxSuppression(bool value);
void set_corner_list(std::list<int> l);
std::list<int> get_corner_List();
void set_fast_xml_doc(QString fn);
void set_fast_xml_doc(QDomDocument doc);
void set_fast_xmlfilename(QString fileName);
QDomDocument get_fast_xml_doc();
//SIFT
void set_sift_list(std::list<int> l);
void set_sift_xml_doc(QString filename);
void set_sift_xml_doc(QDomDocument doc);
void set_sift_threshold(double t);
void set_sift_edgeThreshold(double t);
void set_sift_magnification(double m);
void set_sift_isNormalized(bool b);
void set_sift_recalculateAngle(bool b);
void set_sift_nOctaves(int value);
void set_sift_nOctaveLayers(int value);
void set_sift_firstOctave(int value);
void set_sift_angleMode(int value);
void set_sift_xmlfilename(QString fileName);
std::list<int> get_sift_list(){ return sift_list;}
```

```
QDomDocument get_sift_xml_doc(){ return sift_xml_doc;}
double get_sift_threshold(){return sift_threshold;}
double get_sift_edgeThreshold(){return sift_edgethreshold;}
double get_sift_magnification(){return sift_magnification;}
bool get_sift_isNormalized(){return sift_isNormalized;}
bool get_sift_recalculateAngle(){return sift_recalculateAngle;}
int get_sift_nOctaves(){return sift_nOctaves;}
int get_sift_nOctaveLayers(){return sift_nOctaveLayers;}
int get_sift_firstOctave(){return sift_firstOctave;}
int get_sift_angleMode(){return sift_angleMode;}
QString get_sift_xmlfilename(){return xml_sift_filename;}
//SURF
void set_surf_list(std::list<int> l);
void set_surf_xml_doc(QString filename);
void set_surf_xml_doc(QDomDocument doc);
void set_surf_hessian_threshold(double t);
void set_surf_nOctaves(int value);
void set_surf_nOctaveLayers(int value);
void set_surf_upright(bool value);
void set_surf_extended(bool value);
void set_surf_xmlfilename(QString fileName);
std::list<int> get_surf_list(){ return surf_list;}
QDomDocument get_surf_xml_doc(){ return surf_xml_doc;}
double get_surf_hessian_threshold(){return surf_hessian_threshold;}
int get_surf_nOctaves(){return surf_nOctaves;}
int get_surf_nOctaveLayers(){return surf_nOctaveLayers;}
bool get_surf_upright(){return surf_upright;}
bool get_surf_extended(){return surf_extended;}
QString get_surf_xmlfilename(){return xml_surf_filename;}
// FAST
QDomDocument fast_xml_doc;
QString xml_fast_filename;
int fast_threshold;
bool fast_nonmaxSuppression;
std::list<int> corner_list;
// SIFT
QDomDocument sift_xml_doc;
std::list<int> sift_list;
QString xml_sift_filename;
double sift_threshold;
double sift_edgethreshold;
double sift_magnification;
bool sift_isNormalized;
bool sift_recalculateAngle;
int sift_nOctaves;
int sift_nOctaveLayers;
int sift_firstOctave;
int sift_angleMode;
// SURF
QDomDocument surf_xml_doc;
std::list<int> surf_list;
QString xml_surf_filename;
double surf_hessian_threshold;
int surf_nOctaves;
int surf_nOctaveLayers;
bool surf_upright;
bool surf_extended;
```

La classe *VideoTools* è il core delle funzionalità degli strumenti. I suoi metodi eseguono le operazioni sul video e per questo richiedono una più dettagliata descrizione.

```
void analyzeFAST();
void analyzeSIFT();
void analyzeSURF();
```

Il metodo *analyzeFAST* avvia l'analisi dell'intero video caricato e la creazione del file XML con la feature FAST. Il metodo *analyzeSIFT* avvia l'analisi mediante SIFT mentre *analyzeSURF* avvia l'analisi con SURF..

```
void showFAST(IplImage *img, int pos);
void showSIFT(cv::Mat *img, int pos);
void showSURF(cv::Mat *img, int pos);
```

Il metodo *showFAST* visualizza i corner del frame *img*, che rappresenta il frame di indice *pos* all'interno del video. Lo stesso compito hanno i metodi *showSIFT* e *showSURF* che a differenza del primo ricevono il frame come matrice.

```
CvScalar green;
CvScalar BLUE;
CvScalar FUCSIA;
```

I tre scalari rappresentano le terne RGB dei colori verde (0, 255, 0), blu (0, 0, 255) e fucsia (255, 0, 255), usati per la visualizzazione dei punti FAST, SIFT e SURF rispettivamente. Lo scalare *green* è stato ripreso dall'implementazione di un tool precedentemente inserito in CVideoPlayer.

La classe *XmlWrapper* rapprensenta l'oggetto a cui è demandata la gestione del file XML. A causa della grande dimensione dei file prodotti dalle feature FAST, SIFT e SURF, sono stati aggiunti alla classe molti metodi per la scrittura in modalità "*append*" e funzioni esterne all'oggetto per la lettura "*frame-by-frame*".
La modalità di scrittura **append** scrive singoli nodi frame sul file xml permettendo di tenere in memoria un solo nodo del DomDocument e riducendo la memoria locale utilizzata.
La modalità di lettura **frame-by-frame** premette di leggere singoli frame da un file xml, evitando così di dover importare interi documenti xml di dimensioni proibitive da gestire in memoria locale

Elenco brevemente le costanti aggiunte:

```
#define MAX_FAST_FILE_SIZE 52428800
#define MAX_SIFT_FILE_SIZE 52428800
#define MAX_SURF_FILE_SIZE 52428800

// costanti per FAST
static const char * xml_fast_doctype;
static const char * xml_fast_corner;
static const char * xml_fast_size;
static const char * xml_angle;
static const char * xml_response;
static const char * xml_octave;
static const char * xml_class_id;
// costanti per SIFT
static const char * sift;
static const char * xml_sift_doctype;
static const char * xml_sift_point;
static const char * xml_sift_descriptors;
// costanti per SURF
static const char * surf;
static const char * xml_surf_doctype;
static const char * xml_surf_point;
static const char * xml_surf_descriptors;
```

Elenco brevemente le funzioni delle quali è stato fatto un overload per gestire tipi di dati diversi (come float o tabelle hash con valori multipli):

```
void addNode(QString nodeName, QHash<QString, float> *attributes);
void addNode(QString nodeName, QMultiHash<QString, float> *attributes);
```

I metodi per la lettura dei keypoint di un frame, avendo in memoria l'intero DomDocument, sono i seguenti:

```
static std::vector<ROW_VEC_F> getCornerAtFrame(int pos);
static std::vector<ROW_VEC_F> getPointAtFrame(int pos, QString feature="sift");
```

Il metodo *getCornerAtFrame* ritorna un vettore contenente gli angoli letti dal frame di indice *pos*.
Il metodo *getPointAtFrame* ritorna i keypoint letti dal frame *pos*. Il metodo può essere usato sia per SIFT che per SURF, basta impostare la stringa *feature*.

I metodi per la lettura *frame-by-frame* sono i seguenti:

```
static std::vector<ROW_VEC_F> getCornerAtFrame(QDomDocument frame_doc, int pos);
static std::vector<ROW_VEC_F> getPointAtFrame(int pos, QDomDocument frame_doc,
  Qstring feature="sift");
```

I metodi ritornano i vettori dei keypoint riscontrati nel frame, rappresentato in memoria dal DomDocument *frame_doc*. Questo DomDocument conterrà solo il frame richiesto

Le funzioni esterne alla classe ma utilizzate all'interno dei metodi precedentemente elencati sono:

```
QDomElement parse_fast_corner(QString text);
QDomElement parse_sift_point(QString text);
QDomElement parse_surf_point(QString text);
QDomDocument read_frame_from_file(int num, QString fileName);
std::list<int> read_list_from_file(QString fileName);
bool is_frame_in_list(std::list<int> frame_list, int num);
```

Le funzioni *parse_fast_corner, parse_sift_point, parse_surf_point* si occupano di convertire la stringa *text* contenente il nodo del keypoint in un elemento per il DomDocument.
La funzione *read_frame_from_file* ritorna un *QDomDocument* contenente un unico nodo <frame> riguardante il frame di indice *pos* all'interno del file *filename.*
La funzione *read_list_from_file* ritorna la lista degli indici dei frame all'interno del file *filename*.
La funzione *is_frame_in_list* verifica se l'indice *num* si trova all'interno della lista *frame_list* e nel caso positivo ritorna il booleano *true,* altrimenti *false.*

I metodi per la scrittura in modalità *append* sono i seguenti:

```
void writeToFile_append(QString fileName);
void writeToFile_initialize(QString fileName, QString xml_type);
void writeToFile_finalize(QString fileName, QString xml_type);
```

La funzione *writeToFile_append* scrive l'unico nodo frame del DomDocument sul file indicato da *fileName* aperto in modalità *append.*
La funzione *writeToFile_initialize* inizializza il file indicato da *fileName* con il document type e i tag di root necessari alla feature indicata da *xml_type*.
La funzione *writeToFile_finalize* finalizza il file chiudendo i tag di root della feature *xml_type.*
Attenzione: sia nella funzione di inizializzazione che di finalizzazione, la stringa *xml_type* deve essere la medesima altrimenti i tag non risulteranno chiusi correttamente.

Bibliografia

Edward Rosten, Tom Drummond, "*Machine learning for high-speed corner detection"*, 2006.
David G. Lowe, *"Object Recognition from Local Scale-Invariant Features"*,1999.
David G. Lowe, *"Distinctive Image Features from Scale-Invariant Keypoints "*, 2004.
Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, *"Speeded-Up Robust Features (SURF)"*, 2008
Documentazione di OpenCV, http://opencv.willowgarage.com/documentation/cpp/index.html