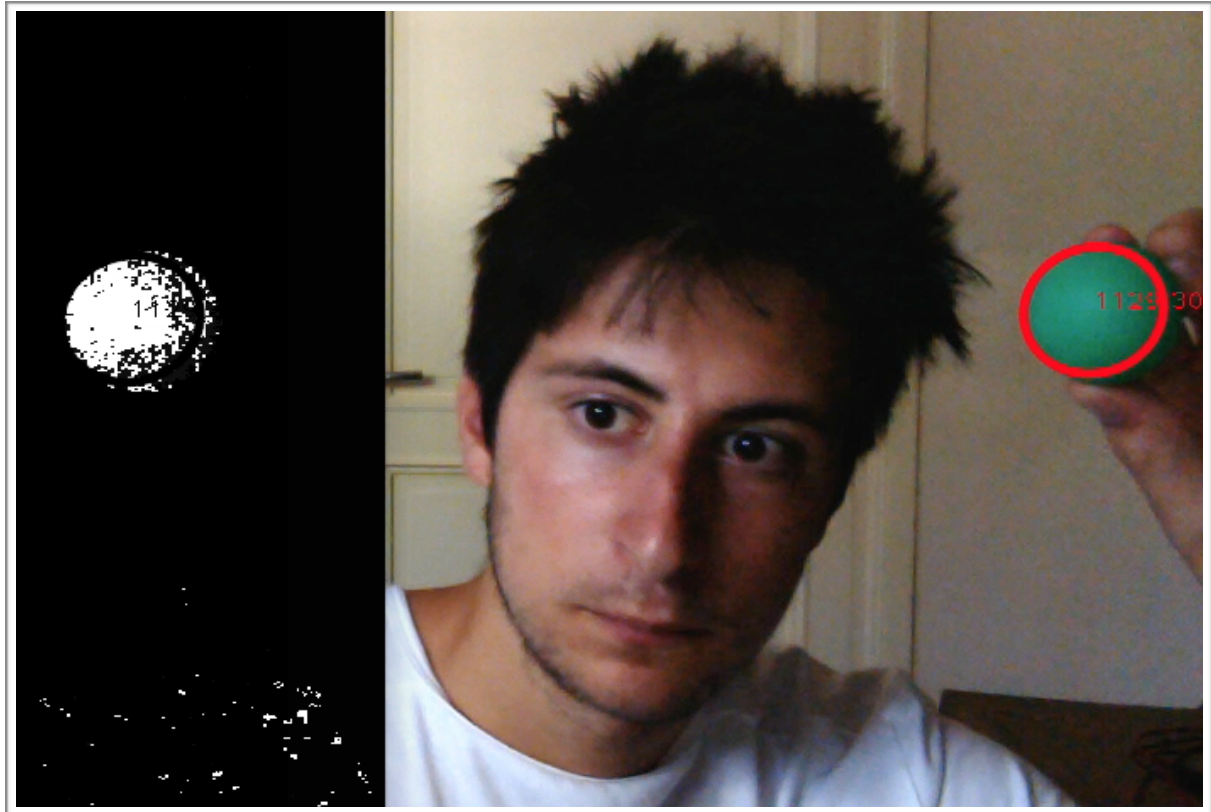


# Real Time Tracking



Laureando: Lorenzo Chelini

September 2015

# Introduction

*What is Tracking ?*

*Application context*

*Goals*

## Three ways of tracking

*Template Matching*

*Tracking based on Colors*

*Blob detection*

## Tracking Algorithm

*Library - OpenCV*

*Code - Object Detection With Colors*

*MeanShift and CamShift Tracking*

## Results

*Test*

*Obtained results*

# Introduction

## What Is Tracking

We talk about tracking when we are considering the ability of an elaborator, in the simplest way a computer, to hook and follow a specific object, which is contained in a sequence of images captured by a camera at a specific frame rate. Some difficulties to be considered rise from the definition of tracking that we should consider.

First of all the environment which the object is immerse into is a dynamic one, it means that we should consider also the changes in the weather condition and in the illumination as well.

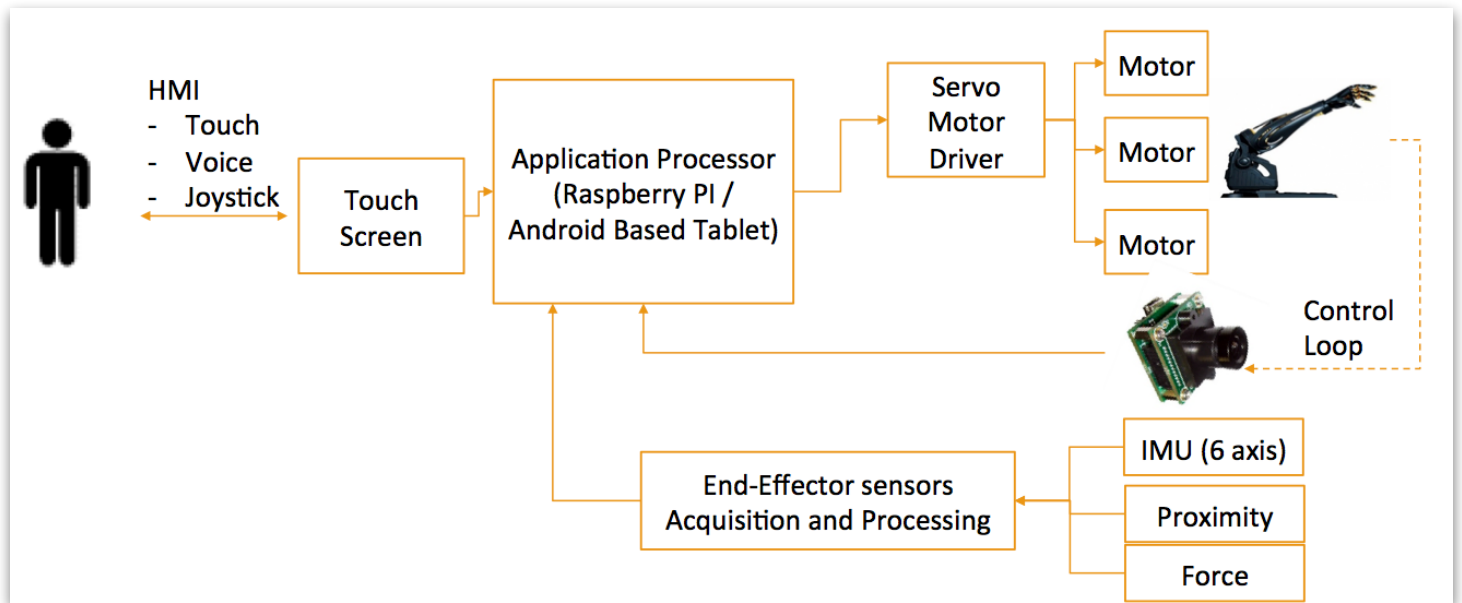
Secondly we should consider that in the environment there could be also inanimate or animated objects that may move around or may be in a slowly transformation. Therefore summarizing (briefly) the difficulties that a tracking algorithm has to face:

- Occlusion caused by objects that are between the target and the camera.
- The changes sense by the camera about the tracking object for example: illumination or pose changes.
- Objects that are similar in shape or dimension to the target may mislead the algorithm.

So, tracking algorithms or in general computer vision real-time systems need a lot of elaboration, that was unthinkable only few years ago: we must only think that this same power elaboration for the human brain keep 60 billion neurons busy. However, nowadays, the progress in technology has mitigated the problem above-mentioned and more and more hardware can solve these problems, with real-time

constrains, in a short frame time. Moreover a lot of techniques and dedicated library has been developed, OpenCv is an example.

# Application Context



The thesis has been written for RAPID project, RAPID stand for **R**obotic **A**rm empowering **P**eople w**I**th **D**isabilities

# Goals

The aim of the thesis is to set up real-time algorithms optimized for a mobile camera. In the next few paragraphs we will also introduce three different techniques for tracking and we will try to understand the pros and cons for each of them. This allows, in future works, to build up much stronger algorithms. Another aspect that requires attention is the rules or methods in order to detect automatically the target loss. So the main characteristic that these types of algorithm should have is a low CPU overhead in order to be executed on dedicated hardware.

# Three ways of tracking

## Template Matching



Template Matching is a high-level machine vision technique that identifies the different parts of an image on the image itself that match a predefined template. Template matching techniques are flexible and relatively straightforward to use, which makes them one of the most important and popular method of objects localization. However their application is limited mostly by the available computation power, as identification of big and complex templates can be time-consuming. Template Matching techniques are expected to address the following need: Once we have provided a reference image of an object (the template image or patch) and an image to be inspected (the input image), we want to identify all input image locations at which the object from the template image is present. To identify the matching area, we have to compare the template image against the source image by sliding it. “By sliding”, means that we move the patch one pixel at a time (left to right, up to down). At each location(x,y), it is calculated a metric that represents how “good” or “bad” the match at that location is. So actually what we do is very simple: we position the template over the image at every possible location, and each time we compute some numeric measures of similarity between the template and the image segment it currently overlaps with. Finally we identify the positions that yield the best similarity measures. One of the sub

problems that occur in the specification above is calculating the similarity measure of the aligned template image and the overlapped segment of the input image, which is equivalent to calculating a similarity measure of two images of equal dimensions. This is a classical task: for this reason OpenCv (the library used in this thesis) makes available some methods which are:

a. `method=CV_TM_SQDIFF`

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

b. `method=CV_TM_SQDIFF_NORMED`

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

c. `method=CV_TM_CCORR`

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

d. `method=CV_TM_CCORR_NORMED`

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

e. `method=CV_TM_CCORR_NORMED`

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

where

$$T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')$$

$$I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')$$

f. `method=CV_TM_CCORR_NORMED`

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

The last key part of this process is to identify which are the points that are good enough to be considered actual matches. This can be done following these sample rules:

- We consider matches the points for which the measuring method gave us a higher value than a predefined threshold.



- We consider matches the points for which the measuring method gave us a higher value than the value that the same method gave us for their neighbors.

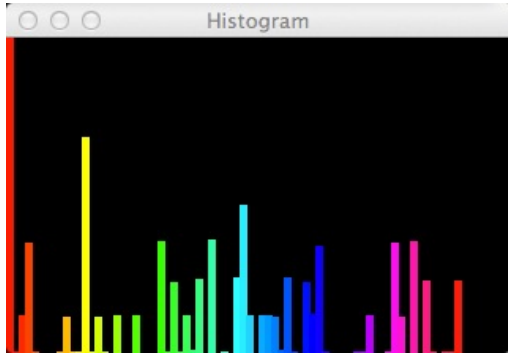
Though the introduced technique was sufficient to solve sample problems, we may notice its important drawbacks:

- Template occurrences have to preserve the orientation of the reference template image.
- The method is inefficient, as calculating the template correlation image for medium to large images is time wasting.

### **Advance template matching technique:**

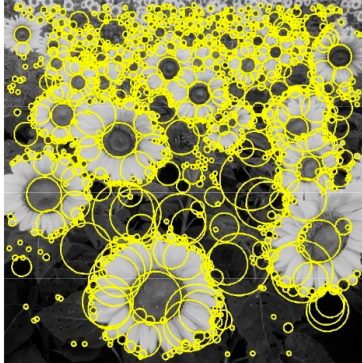
Pyramid Processing is an advanced template matching algorithm that extends the original idea enhancing its efficiency. The Image pyramid is a series of images, each image being a result of downsampling (means scaling down) of the previous element. The important observation is that the template depicted in the reference image usually is still discernible after significant downsampling of the image (though, naturally, fine details are lost in the process). Therefore we can identify match candidates in the downsampled (and therefore much faster to process) image on the highest level of our pyramid, and then repeat the search on the lower levels of the pyramid; each time considering only the template positions that scored high on the previous level. At each level of the pyramid we will need appropriately downsampled picture of the reference template, i.e. both input image pyramid and template image pyramid should be computed.

# Tracking Based On Colors



Tracking based on colors is another popular method of object localization. The main idea consists of using the chromatic characteristics of the target in order to distinguish it from the background. The technique expects to build a histogram, using the chromatic characteristics of the target, to compare with the image. The result is a bunch of pixels that have been located according to the level of correspondence. Moreover modern techniques use, in addition, clustering algorithm in order to identify the area in which the pixels (whose chromatic color is the same as the target) density is maximum. One of the limitation that this type of algorithm has is the fact that the target should have different colors from the background.

# Blob Detection



Blob detection method is aimed at detecting regions in an image that differ in properties, such as for example color or brightness, compared to surrounding regions. So a Blob is a region of an image in which some properties are constant (or approximately constant); all the pixels in a

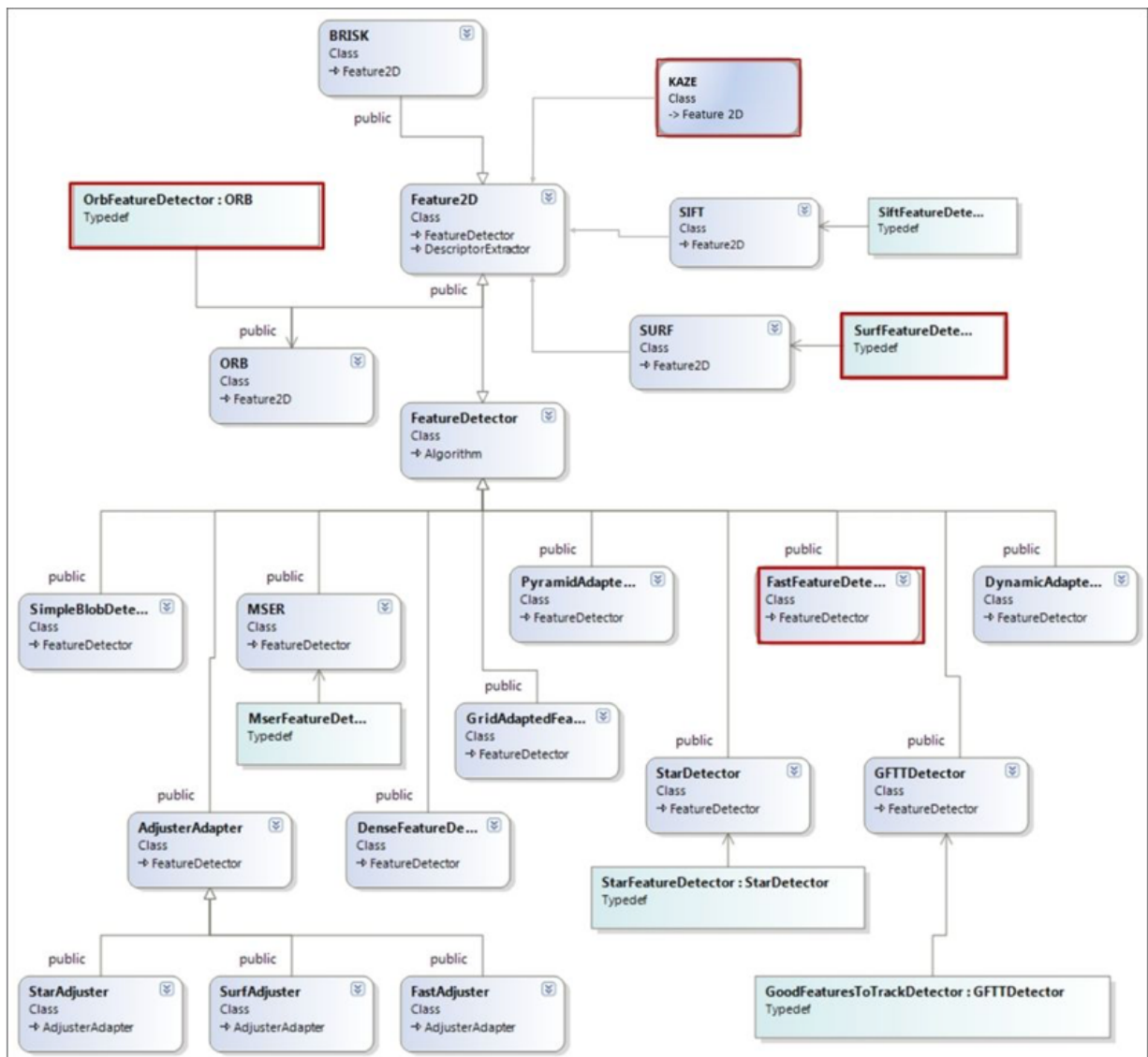
Blob can be considered in some way to be similar to each other. There are two main classes of blob detectors:

- Differential Methods
- Method based on local extrema (or Interest point operators) which are based on finding the local maxima and minima of the function.

In this thesis we will consider only the second one, Interest point operators. In computer vision the concept of interest points also called keypoints or feature points, has been largely used to solve many problems in object recognition. This concept relies on the idea that instead of looking at the image as a whole, it could be advantageous to select some special points in the image and perform a local analysis on them. This approach works well as long as a sufficient number of such points are detected in the image of interest and these points distinguishing. As they are used for analyzing image content, feature points should ideally been detected at the same scene or object location no matter from which viewpoint, scale or orientation the image was taken. View invariance is a very desirable property in image analysis and has been the object of numerous studies. So how extract keypoints from the images? When searching for interesting feature points in images, corner come out as an interesting solution.

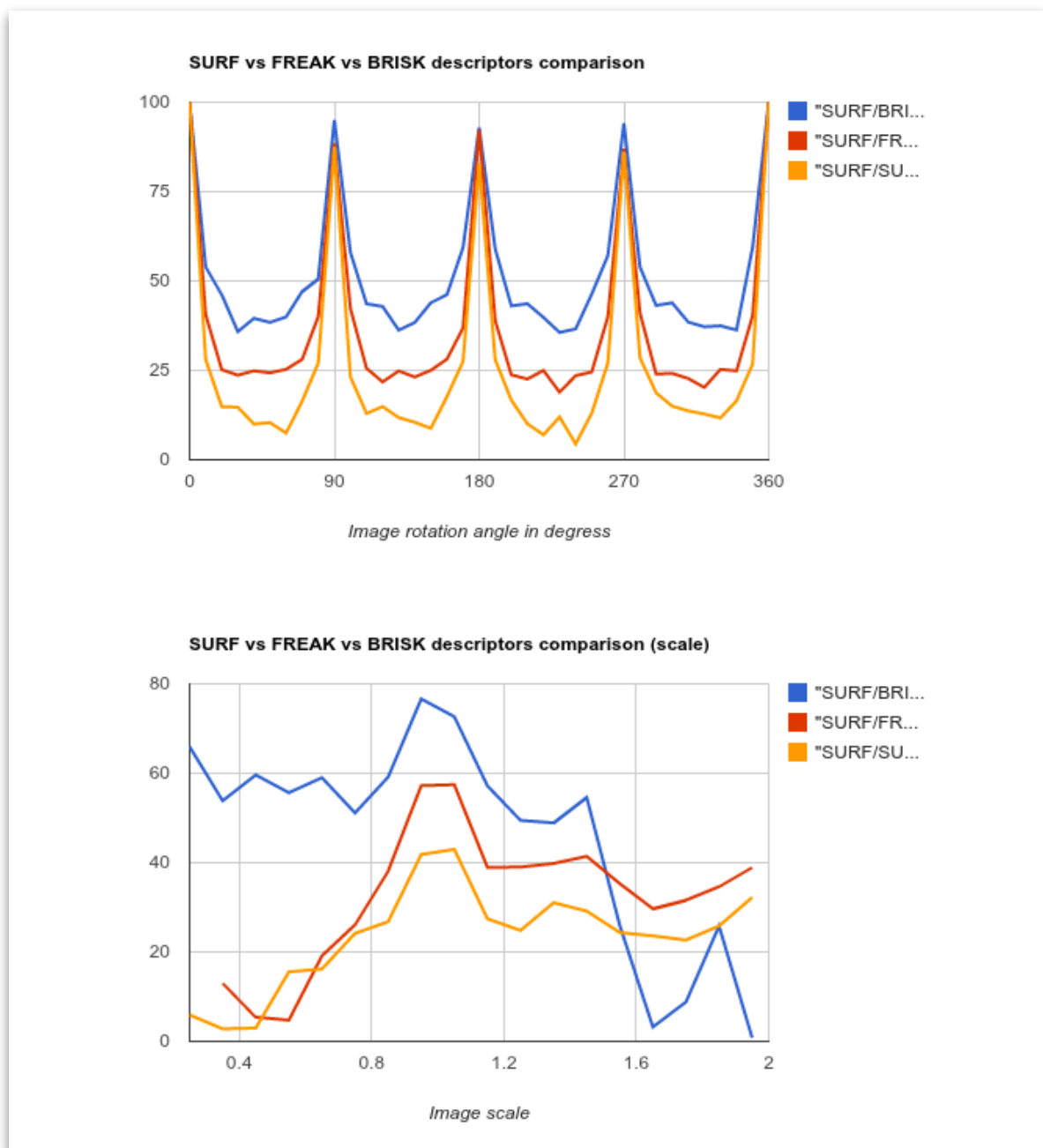
Corner are interesting because they are two dimensional features that can be accurately localized, as they are at the junction of two edges. OpenCV handles several local feature detector implementations thorough the FeatureDetector abstract class and its `Ptr<FeatureDetector>FeatureDetector::create(const string& detectorType)` method or through the algorithm class directly. In the first case, the type of detector is specified (see the following diagrams)

- FAST (FastFeatureDetector): This feature detects corners and blobs
- STAR (StarFeatureDetector): This feature detects edges, corners, and blobs
- SIFT (SiftFeatureDetector): This feature detects corners and blobs (part of the nonfree module)
- SURF (SurfFeatureDetector): This feature detects corners and blobs (part of the nonfree module)
- ORB (OrbFeatureDetector): This feature detects corners and blobs
- BRISK (BRISK): This feature detects corners and blobs
- MSER (MserFeatureDetector): This feature detects blobs
- GFTT (GoodFeaturesToTrackDetector): This feature detects edges and corners
- HARRIS (GoodFeaturesToTrackDetector): This feature detects edges and corners (with the Harris detector enabled)
- Dense (DenseFeatureDetector): This feature detects the features that are distributed densely and regularly on the image
- SimpleBlob (SimpleBlobDetector): This feature detects blobs

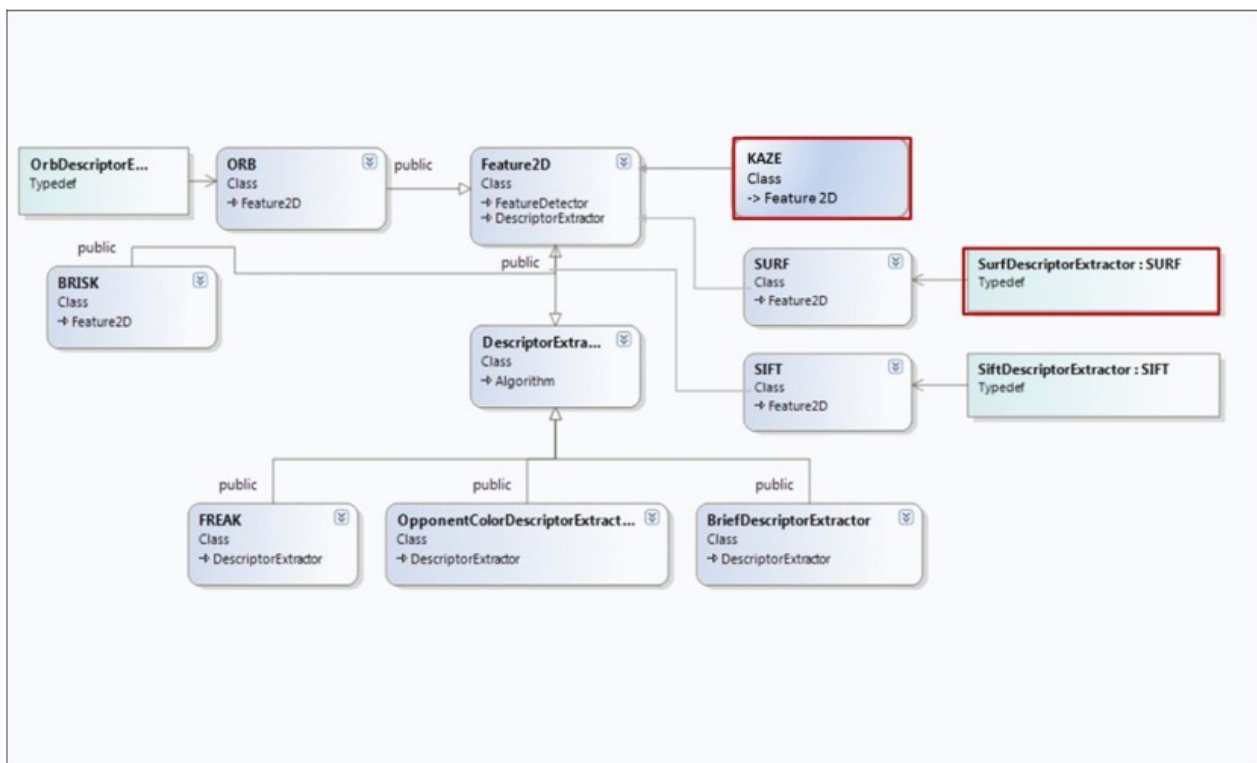


*2D feature detectors in OpenCV*

The best results in pattern detection are achieved if the detector computes keypoint orientation and size. This makes keypoints invariant to rotation and scale. The most famous and robust keypoint detection algorithm are well known: they are used in SIFT and SURF feature detection and descriptor extraction.



Once we have our keypoint detected we have to use descriptor extractors in order to provide a measure and distance function for a small patch around interest point. Therefore, whenever the similarity between two images patches needs to be estimated, we compute their descriptor and measure the distance. In OpenCV the `Ptr<DescriptorExtractor>DescriptorExtractor::create(const String&DescriptorExtractorType)` function creates a new descriptor extractor of the selected type. (see the diagram above)



*2D feature descriptors in OpenCV*

# Tracking Algorithm

## Library - Opencv

OpenCV is an open source (<http://opensource.org>) computer vision library available from <http://SourceForge.net/projects/opencvlibrary>. The library is written in C and C++ and runs under Linux, Windows and Mac OS X. OpenCV was designed for computational efficiency and with strong focus on real-time application. One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including medical imaging, security, user interface and robotics.





# Code - Object Detection With Colors

## Some details:

Requirements needed in order to use this software:

- you should install the last version of OpenCV library (the version used for development is OpenCV 3.0).
- you should have a default camera.

Using this software the user can select and track an object in real-time.

The main idea is illustrated below, once the user selected the area that match a sample of what he or she is looking for, the algorithm start a real time elaboration scanning the frame captured by the default camera (the frame rate can be modified by the user) in order to locate the ROI (Region Of Interest, or in other worlds, the area selected by the user).

There are three main steps:

In the first one the algorithm create, using calcHist function, the histogram of the image, we can thing the histogram as a simple table that gives you the number of pixels that have a given value in an image.

The histogram of a grey-level image will, therefore, have 256 entries (or bins).

Bin 0 gives you the number of pixels that have the value 0, bin 1 gives you the number of pixels that have the value 1, and so on.

Once the histogram of the ROI image has been created, the second step comes into play.

In this step we use, calcBackProject function, so what mean BackProject an histogram ?

Well, BackProjecting a histogram consists of replacing each pixel value in an input image with its corresponding probability value read in the normalized histogram (by normalizing an histogram we obtain a function that gives us the probability that a pixel of a given intensity value belongs to the defined area).

The result of this step is a probability map, with probability belonging to the reference area ranging from bright (low probability) to dark (high probability).

In the third step we use CamShift Algorithm in order to locate the exact position of the object, in other words the result of a histogram back-projection, as we have just said, is a probability map that express the probability that a given piece of image content is found at a specific image location.

So if we know the approximate location of an object in the image, the probability map can be used in order to find the exact position of the object.

The most probable location will be the one that maximizes this probability inside a given window.

In conclusion if we know the approximate location of the object and if we start looking from that point and iteratively move around we should locate the exact position of the object, that stuff is done by the Camshift function.

## Code Structure

### Classe: Camera

```
class camera{
private:

    enum class CameraResolution{
        RES_640x480 = 0,          /**< 640x480 pixel resolution*/
        RES_320x240,             /**< 320x240 pixel resolution*/
        RES_1280x960             /**< 1280x960 pixel resolution*/
    };

    enum class RetType {
        OK = 0,                  /**< NO error*/
        Error,                   /**< Generic error*/
        UnsupportedRes,          /**< Resolution non supported by the camera */
        UnsupportedFrameRate,    /**< Frame Rate non supported by the camera */
        CameraErrorIndex,       /**< Unable to get the camera */
        OutofResError,          /**< Out of Resolution error*/
        ErrorParsing,           /**< Error while reading parameters*/
    };

    CameraResolution camRes;    //camera resolution (read from xml file)

    int frameRate;              //frame rate (read from xml file)
    int cameraIndex;           //camera index (read from xml file)

    cv::Mat frame;             //next frame captured by camera
    cv::VideoCapture video;

    xmlParser* xml;            //parser
};
```

```

    RetType assega(std::string what, std::string value); //assign camera's parameters, called by
setParam()
    bool setCameraRis(); //set camera resolution

public:

    camera(xmlParser xml); //constructor
    bool openCamera(); //open camera
    bool setParam(); //set camera's parameters
    int getFrameRate(); //getFrameRate
    cv::Mat nextFrame(); //get next frame

};

```

## Camera.cpp

**Note:** this method open video stream using the cameraIndex read in the xml file

```

bool camera:: openCamera(){
    video.open(cameraIndex);
    if(!video.isOpened())
        return false;
    return true;
}

```

**Note:** this method return the frame rate, read from the xml file, i want to underline that this method is a public one because is used by ObjectDetector.cpp in particular by the public method StartTracking().

```

int camera::getFrameRate(){
    return frameRate;
}

```

**Note:** this public method used by ObjectDetector.cpp in particular by StartTracking(), return the frame captured by the camera NULL otherwise.

```

cv::Mat camera::nextFrame(){
    cv::Mat frame;

    if (!video.read(frame)) {
        std::cout << "Unable to retrieve frame from video stream." << std::endl;
    }

    if( frame.empty() )
        frame = NULL;
    return frame;
}

```

**Note:** this public method is used in order to read and set camera parameters (note that the method call `assegna(std::string w, std::string v)`)

I use a `std::vector` in order to get the parameters from xml file (see `parse()` for more details) note that before using the `parse` method you have to update the target in the xml file so the correct procedures are:

- 1) `updateTarget`
- 2) `call parse()`

Additionally note that the data are stored in the vector in this way:  
`[name,data] + [name,data] + [].....` and so on.

Tips: this boolean method return false if an error occurred and the program ends, but we can do in this other way : if an error occurred we can set a flag and then, after the opening of the camera we can add method in order to read camera resolution (with `videoCapture.get`) or we can set default values.

```

bool camera:: setParam(){
    std::vector<std::string> list;
    camera::RetType ris;
    xml->updateTarget("camera");
    list = xml->parse();
    if(list.size() == 0){

```

```

    ris = RetType::ErrorParsing;
    return false;
}

for(int i=0; i<list.size(); i = i+2){
    std::string what = list.at(i);
    std::string value = list.at(i+1);
    ris = assegna(what,value);
    switch(ris) {
        case camera::RetType::UnsupportedFrameRate : std::cout << "unsupported frame
rate" << std::endl; return false; break;
        case camera::RetType::CameraErrorIndex : std::cout << "camera index error" <<
std::endl; return false; break;
        case camera::RetType::UnsupportedRes : std::cout << "unsupported resolution" <<
std::endl;return false; break;
        default: break;
    }
}

return true;
}

```

**Note:** this private method called by setParam is used in order to assign the values read from xml file.

if an error has occurred during the reading return the particular type of error that will be catch by setParam().

Moreover note that this method call setCameraRis().

```
camera::RetType camera::assegna(std::string w, std::string v){
```

```
    RetType ris = RetType::OK;
```

```
    const char * what = w.c_str();
```

```
    const char* value = v.c_str();
```

```
    if(std::strcmp(what,"frameRate")==0){
        std::cout << "setto frame rate" << std::endl;
        int fr = std::atoi(value);
        //frame rate realizzato con wiatKey() basta
        //che sia maggiore di 0
        if(fr <= 0){
            std::cout << "frame Rate <= 0" << std::endl;
            ris = RetType::UnsupportedFrameRate;
        }
        frameRate = fr;
        return ris;
    }
```

```
    if(std::strcmp(what,"cameraIndex")==0){
        std::cout << "setto camera Index" << std::endl;
        int ci = std::atoi(value);
        if(ci < 0){
            std::cout << "Camera Index <= 0" << std::endl;
            ris = RetType::CameraErrorIndex;
        }
        cameraIndex = ci;
        return ris;
    }
```

```
    if(std::strcmp(what,"cameraResolution")==0){
        std::cout << "setto risoluzione camera" << std::endl;
        int index = std::atoi(value);
        switch (index) {
            case 0:
                camRes = CameraResolution::RES_320x240;
                if(!setCameraRis())
                    ris = RetType::UnsupportedRes;
        }
```



```
        break;
    case 1:
        camRes = CameraResolution::RES_640x480;
        if(!setCameraRis())
            ris = RetType::UnsupportedRes;
    default:
        camRes = CameraResolution::RES_1280x960;
        if(!setCameraRis())
            ris = RetType::UnsupportedRes;
        break;
    }

    return ris;
}
return ris;
}
```

**Note:** this private method called by `assegna` set the camera resolution using `videoCapture::set` method.

```
bool camera:: setCameraRis(){
    bool width = true;
    bool height = true;

    switch(camRes) {
        case CameraResolution::RES_320x240:
            if(!video.set(CV_CAP_PROP_FRAME_WIDTH,320))
                width = false;
            if(!video.set(CV_CAP_PROP_FRAME_HEIGHT,240))
                height = false;
            break;
        case CameraResolution::RES_640x480:
            if(!video.set(CV_CAP_PROP_FRAME_WIDTH,640))
                width = false;
            if(!video.set(CV_CAP_PROP_FRAME_HEIGHT,480))
                height = false;
            break;
        default:
            if(!video.set(CV_CAP_PROP_FRAME_WIDTH,1280))
                width = false;
            if(!video.set(CV_CAP_PROP_FRAME_HEIGHT,960))
                height = false;
            break;
    }
    return width && height;
}
```

## Classe: XMLparser

Note: here we use rapidXML : <http://rapidxml.sourceforge.net/>

Manual : <http://rapidxml.sourceforge.net/manual.html>

```
class xmlParser{

private:
    std::string file; //target file
    std::string target; //target node

    std::string getTarget(); //get target name

public:
    xmlParser(std::string); //constructor
    void updateTarget(std::string); //modify target
    std::vector<std::string> parse(); //parsing
};
```

### XMLparser.cpp

Note: this method update the target node.

```
void xmlParser::updateTarget(std::string update){
    target = update;
}
```

Note: this method return the target node.

```
std::string xmlParser::getTarget(){
    return target;
}
```

**Note:** this method return a vector `<std::String >` filled with data.

The data are filled in this way :

`[name + data],[name + data],...`,

First of all the method open the target file or exit if an error has occurred during that operation, then the method catch the root node in the xml file identified by target and lookup for all of his children and get their names and values.

```
std::vector<std::string> xmlParser::parse(){
    std::vector<std::string> list;
    list.clear();

    std::string target = getTarget();
    const char * w = target.c_str();

    std::cout << "Parsing my file..." << std::endl;
    rapidxml::xml_document<> doc;
    rapidxml::xml_node<> * root_node;

    std::ifstream theFile ("../parametri.xml");
    if(!theFile.good()){
        std::cout << "File not found" << std::endl;
        exit(-1);
    }

    std::vector<char> buffer((std::istreambuf_iterator<char>(theFile),
std::istreambuf_iterator<char>()));
    buffer.push_back('\0');

    doc.parse<0>(&buffer[0]);

    root_node = doc.first_node("parametri");

    rapidxml::xml_node<> * object_node = root_node->first_node(w);

    for(rapidxml::xml_node<>* child = object_node->first_node(); child; child = child-
>next_sibling()){
        list.push_back(child->first_attribute("name")->value());
    }
}
```

```

    list.push_back(child->value());
}

return list;
}

```

## Classe: ObjectDetector

```

class ObjectDetector{
private:

enum class RetType {
    OK = 0,          /**< NO error*/
    Error,          /**< Generic error*/
    ErrorParsing,   /**< error while reading parameters*/
};

bool selectObject;
int trackObject;
std::string windowName;

cv::Mat image;
cv::Mat hsv;
cv::Mat hue;
cv::Mat mask;
cv::Mat hist;
cv::Mat backproj;
cv::Rect trackWindow;

xmlParser* xml;
camera* cam;

struct targetCoordinates{

```

```

    cv::Point origin;
    cv::Point lastCenter;
    cv::Rect selection;
}tc;

struct hsvParameters{
    int hsize; //number of bins
    float hranges[2]; // pixel value ranges
    const float* phranges = hranges;
}hsvParam;

struct targetArea{
    int counter;
    cv::Size2f mArea;
    float h;
    float w;
}ta;

bool readParam(std::string w);
void setParam(std::string w, std::string v);
void openWindow();
void onMouse(int event, int x, int y, int flags );
bool elaborate();
double norm_L2(const cv::Point &x , const cv::Point &y);

public:
    ObjectDetector(xmlParser &xml, camera &c);
    void startTracking();
    static void onMouseWrapper( int event, int x, int y, int flags, void* that);

};

```

## ObjectDetector.cpp

**Note:** this method look quite similar to `setParam()` in `Camera.cpp`, so nothing new must be reported.

this method is called by the constructor.

```
bool ObjectDetector::readParam(std::string w){

    std::vector<std::string> list;
    xml->updateTarget(w);
    list = xml->parse();
    if(list.size() == 0){
        return false;
    }
    for(int i=0; i<list.size(); i = i+2){
        std::string what = list.at(i);
        std::string value = list.at(i+1);
        setParam(what,value);
    }

    return true;

}
```

**Note:** this method look quite similar to `assegna()` in `Camera.cpp`, so nothing new must be reported.

this method is called by the `readParam()`.

```
void ObjectDetector::setParam(std::string w, std::string v){

    const char * what = w.c_str();
    const char* value = v.c_str();

    if(std::strcmp(what,"hsize")==0){
        std::cout << "setto hsize" << std::endl;
        hsvParam.hsize = std::atoi(value);
    }

    if(std::strcmp(what,"hranges0")==0){
```

```

        std::cout << "setto hranges first param" << std::endl;
        hsvParam.hranges[0] = std::atoi(value);
    }

    if(std::strcmp(what,"hranges1")==0){
        std::cout << "setto hranges second param" << std::endl;
        hsvParam.hranges[1] = std::atoi(value);
    }

    if(std::strcmp(what,"windowName")==0){
        std::cout << "setto windowName" << std::endl;
        windowName = value;
    }
}

```

**Note:** this method is called by StartTracking().

This method create a new window and set on this window the MouseCallback handler.

WindowName represent the window's name and it is read from the xml file.

```

void ObjectDetector::openWindow(){
    cv::namedWindow(windowName, cv::WINDOW_AUTOSIZE);
    cv::setMouseCallback(windowName, ObjectDetector::onMouseWrapper,this);
}

```

**Note:** this method has to be added because the native function cv::setMouseCallback wants a static method to be called.

```

void ObjectDetector::openWindow(){
    cv::namedWindow(windowName, cv::WINDOW_AUTOSIZE);
    cv::setMouseCallback(windowName, ObjectDetector::onMouseWrapper,this);
}

```



## Note:

```
void ObjectDetector::onMouse(int event, int x, int y, int flags){
    if(selectObject)
    {
        tc.selection.x = MIN(x, tc.origin.x);
        tc.selection.y = MIN(y, tc.origin.y);
        tc.selection.width = std::abs(x - tc.origin.x);
        tc.selection.height = std::abs(y - tc.origin.y);
        tc.selection &= cv::Rect(0, 0, image.cols, image.rows);
    }

    switch(event)
    {
        case cv::EVENT_LBUTTONDOWN:
            tc.origin = cv::Point(x,y);
            tc.selection = cv::Rect(x,y,0,0);
            selectObject = true;
            break;
        case cv::EVENT_LBUTTONUP:
            selectObject = false;
            tc.lastCenter = tc.origin;
            if( tc.selection.width > 0 && tc.selection.height > 0 ){
                trackObject = -1;
            }
            break;
    }
}
```

**Note:** this public method first of all calls the readParam function in order to read from the xml file interface parameters such as window's name for example.

Secondly it calls openWindow() method and openCamera() (see these methods for more details)

The for cycle is used to catch new frame and elaborate it within a specific frame rate time forced by waitKey() method.

```
void ObjectDetector::startTracking(){

    readParam("UserInterface");

    openWindow();

    cam->openCamera();

    for(;;){

        image = cam->nextFrame();

        if(image.empty()){
            std::cout << image.empty() << std::endl;
            break;
        }

        cvtColor(image, hsv, cv::COLOR_BGR2HSV);

        cv::Rect trackWindow;

        if(!elaborate())
            return;

        if( selectObject && tc.selection.width > 0 && tc.selection.height > 0 )
        {
            cv::Mat roi(image, tc.selection);
            bitwise_not(roi, roi);
        }
    }
}
```

```

    }

    imshow(windowName, image );

    char c = (char)cv::waitKey(cam->getFrameRate());
    if( c == 27 )
        break;

    }

}

```

**Note:**This private method elaborates the image.

Let's analyze the first if.

In the first part : the method prepares the hsv parameters for the elaboration and also if `trackObject < 0`, (it means that the roi has been selected by the user) calculates the histogram.

In the second part the method calls CamShift function and evaluates two conditions:

The first condition compares the height and the width of the tracked object with height and width approximation that has been made in the first 50 iterations.

The purpose of this condition is to control the shape of the tracked object if this shape change too much it means that i have lost the object.

The second condition calculate the distance between the last center of the tracked object and the new one, if this distance is bigger than 100 it means that probably i have lost the tracked object.

Finally if the conditions have been respected the method updates the position of the object.

```

bool ObjectDetector::elaborate(){

    if( trackObject )
    {
        //se trackobject diverso da zero
        cv::inRange(hsv, cv::Scalar(0, 30, 10),
            cv::Scalar(180, 256, 256), mask);
        int ch[] = {0, 0};
        hue.create(hsv.size(), hsv.depth());
        mixChannels(&hsv, 1, &hue, 1, ch, 1);
        //imshow( "mix", hsv );

        if( trackObject < 0 )
        {
            //caso di trackobject = -1, settato in questo
            //modo dopo avere selezionato il Roi
            //genero histogram
            cv::Mat roi(hue, tc.selection), maskroi(mask, tc.selection);
            calcHist(&roi, 1, 0, maskroi, hist, 1, &hsvParam.hsize, &hsvParam.phranges);
            normalize(hist, hist, 0, 255, cv::NORM_MINMAX);
            trackWindow = tc.selection;
            trackObject = 1;
        }

        calcBackProject(&hue, 1, 0, hist, backproj, &hsvParam.phranges);
        backproj &= mask;
        cv::RotatedRect trackBox = CamShift(backproj, trackWindow,
            cv::TermCriteria( cv::TermCriteria::EPS |
                cv::TermCriteria::COUNT, 10, 1 ));

        if(ta.counter < 50){
            ta.mArea += trackBox.size;
            ta.counter ++;
            ta.h = ta.mArea.height/ta.counter;
            ta.w = ta.mArea.width/ta.counter;
        }

        if(abs(trackBox.size.width - ta.w) > 50 || (abs(trackBox.size.height - ta.h) > 50)){
            std::cout << "lost track for width or height" << std::endl;
            std::cout << abs(trackBox.size.width - ta.w) << std::endl;
            std::cout << abs(trackBox.size.height - ta.h) << std::endl;
        }
    }
}

```

```

        return false;
    }

    if(norm_L2(trackBox.center, tc.lastCenter)>100){
        std::cout << "lost Track for center" << std::endl;
        std::cout << norm_L2(trackBox.center, tc.lastCenter) << std::endl;
        return false;
    }

    //update
    tc.lastCenter = trackBox.center;

    ellipse( image, trackBox, cv::Scalar(0,0,255), 3, cv::LINE_AA );

}

return true;
}

```

**Note:** this method calculate the distance between two points.

```

double ObjectDetector:: norm_L2(const cv::Point &x, const cv::Point &y)
{
    return std::sqrt(((double)((x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y)));
}

```

## XML file

```
<?xml version="1.0" encoding="utf-8"?>
<parametri>
  <camera name="iSight">
    <frameRate name="frameRate">1</frameRate>
    <cameraIndex name="cameraIndex">0</cameraIndex>
    <camRes name="cameraResolution">1</camRes>
  </camera>
  <hsv name="hsvParameters">
    <hsize name="hsize">16</hsize>
    <hranges0 name="hranges0">0</hranges0>
    <hranges1 name="hranges1">180</hranges1>
  </hsv>
  <UserInterface name="userInterface">
    <window name="windowName">OpenCV</window>
  </UserInterface>
</parametri>

<!-- il documento è stato validato: http://www.w3schools.com/xml/xml\_validator.asp -->
```

## XSD schema

(generated using this tool: <http://www.freeformatter.com/xsd-generator.html>)

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="parametri">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="camera">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="frameRate">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:byte">
                      <xs:attribute type="xs:string" name="name"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            <xs:element name="cameraIndex">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:byte">
                    <xs:attribute type="xs:string" name="name"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          <xs:element name="camRes">
            <xs:complexType>
```

```

    <xs:simpleContent>
      <xs:extension base="xs:byte">
        <xs:attribute type="xs:string" name="name"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute type="xs:string" name="name"/>
</xs:complexType>
</xs:element>
<xs:element name="hsv">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="hsize">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:byte">
              <xs:attribute type="xs:string" name="name"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="hranges0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:byte">
              <xs:attribute type="xs:string" name="name"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="hranges1">

```



```

    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:short">
          <xs:attribute type="xs:string" name="name"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:sequence>
  <xs:attribute type="xs:string" name="name"/>
</xs:complexType>
</xs:element>
<xs:element name="UserInterface">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="window">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute type="xs:string" name="name"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute type="xs:string" name="name"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

# Meanshift And Camshift Tracking

The meanShift algorithm is a robust method of finding local extrema in the density distribution of a data set.

The meanShift algorithm runs as follow:

- Choose a search window : its initial location, its type, its shape and its size.
- Compute the window's center of mass.
- Center the window at the center of mass
- Return to step 2 until the window stops moving.

Start with a kernel  $K(\mathbf{X} - \mathbf{X}_i) = ck \left( \left\| \frac{\mathbf{X} - \mathbf{X}_i}{b} \right\|^2 \right)$  approximation of a probability

distribution  $P(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n K(\mathbf{x} - \mathbf{x}_i)$ . Focus on the gradient  $\nabla P(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \nabla K(\mathbf{x} - \mathbf{x}_i)$

Let:  $\mathbf{g}(\mathbf{x}) = -k'(\mathbf{x})$ , the derivative of the kernel and we get:

$$\nabla P(\mathbf{x}) = \frac{c}{n} \sum_{i=1}^n \nabla K_i = \frac{c}{n} \left[ \sum_{i=1}^n g_i \left( \frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{b} \right) \right]$$

The diagram illustrates the mean-shift algorithm's core components. It shows a scatter plot of data points with a search window (a circle) centered at a point  $\mathbf{x}$ . A dashed box highlights the kernel function components: the numerator is the sum of  $x_i g_i$  and the denominator is the sum of  $g_i$ , both over  $i$  from 1 to  $n$ . The window size  $b$  is indicated. A dashed arrow labeled 'Meanshift vector' points from the current center  $\mathbf{x}$  towards the center of mass of the data points within the window.

Figure 10-11. Mean-shift equations and their meaning

The algorithm can be used for visual tracking. In this case, the color histogram of the tracked object is used to compute the confidence map.

The simplest of such algorithm would create a confidence map in the new image based on the object histogram taken from the previous image and MeanShift is used to find the peak of the confidence map near the object's previous position.

The confidence map is a probability density function on the new image assigning each pixel of the new image a probability, which is the probability of the pixel color of the pixel color occurring in the object in the previous image.

### CamShift

A related algorithm is the CamShift tracker. It differs from the MeanShift in that the search window adjusts itself in size.

# Results

## Test

**Camera Used:** ISight di Apple.

### Details:

Letture scheda	
Masterizzazione disco	
Memoria	
NVMeExpress	
PCI	
SAS	
SATA/SATA Express	
SCSI parallelo	
SPI	
Schede Ethernet	
Stampanti	
Thunderbolt	
<b>USB</b>	Posizione controller host: USB integrata Driver controller host: AppleUSBHCI ID dispositivo PCI: 0x1c2d ID revisione PCI: 0x0005 ID fornitore PCI: 0x8086 Numero bus: 0xfa
Network	
Firewall	
Posizioni	
Volumi	
WWAN	
Wi-Fi	
Software	
Accessibilità	
Applicazioni	
Client gestito	
Componenti	
Elementi di avvio	
Estensioni	
Font	
Framework	
Installazioni	
Log	
Pannelli Preferenze	
Profili	
Servizi di sincronizza...	
Software disabilitato	
Software stampante	
Sviluppatore	

**Fotocamera HD FaceTime (integrata):**


ID prodotto:	0x850b
ID fornitore:	0x05ac (Apple Inc.)
Versione:	7.55
Numero di serie:	0000000000000000
Velocità:	Fino a 480 Mb/sec
Produttore:	Apple Inc.
ID posizione:	0xfa200000 / 3
Corrente disponibile (mA):	500
Corrente necessaria (mA):	500
Integrato:	Sì

**Hub:**

ID prodotto:	0x2514
ID fornitore:	0x0424 (SMSC)
Versione:	0.03
Velocità:	Fino a 480 Mb/sec
ID posizione:	0xfa100000 / 2
Corrente disponibile (mA):	500
Corrente necessaria (mA):	2
Integrato:	Sì

**BRCM2046 Hub:**

ID prodotto:	0x4500
ID fornitore:	0x0a5c (Broadcom Corp.)
Versione:	1.00
Velocità:	Fino a 12 Mb/sec
Produttore:	Apple Inc.
ID posizione:	0xfa110000 / 4
Corrente disponibile (mA):	500
Corrente necessaria (mA):	0
Integrato:	Sì



The application has been tested during the planning and implementing phase.

We have created a lots of different situation in order to understand the weaknesses of the algorithm.

The parameters, thorough which the algorithm has been tested follow the requires:

- real-time constrains.
- the algorithm should allows the user to select all kind of objects.
- the tracked object could move around either the camera.

Final Considerations:

In a dynamic context the changes in illumination are, without any dubs, the major problems. They can causes changing in: colors, sharpness and also create shadows.

For this reason we have done a few tests changing the illumination and appeared that if the light changing are quite limited the algorithm can still track the objects.