

# Robust tracking of input props with webcams

Supervisor: Prof. dr. ir. Robert van Liere <sup>1</sup>  
Student: Nguyen Duc Quoc Anh <sup>2</sup>

Amsterdam 2005

<sup>1</sup> Centrum voor Wiskunde en Informatica

<sup>2</sup> University of Amsterdam



## ACKNOWLEDGEMENT

Two years have passed so quickly and I am now at the end of the master program at the University of Amsterdam. It is difficult to have to say goodbye to a place where you have been used to. Looking back upon the time when I first came here, I realize that I have been a lot more mature, both in professional matters and ways of thinking. Back at that time, I was not quite sure what would become of me. Now at the end of the program I feel like I am a new person, quite confident about what I am going to do in the future. As students came from different backgrounds, it was actually a hard time at the beginning since we had too much to cope with but then I began to get on and learn things which were quite new and interesting to me. The program has been a nice experience, indeed. Having said that, I would like to take this chance to express my gratefulness to the University and the Faculty of Computer Science. Thank you for the nice program which you have offered us.

I am so thankful to my daily supervisor, Prof. Robert van Liere, who has provided me with the chance to do the research at the CWI. During the time of the project, I have been taught a great deal on how to carry out scientific work. I feel so glad to have got to know this interesting project and to have had the chance to be a small part of it. Getting to know the CWI is quite interesting. The working environment, the people... all are a good combination of discipline and well being. It has been so comfortable to work here at the CWI.

I thank you all for the nice time living and studying here in the Netherlands. It is going to be one of the best experiences in my life.

Last but not least, I would like to give my deepest gratitude to my parents, who gave me this life and who always have faith in me and my abilities.

Amsterdam, September 2005  
Nguyen Duc Quoc Anh

## TABLE OF CONTENTS

Chapter I INTRODUCTION	4
1. The Personal Space Station	5
2. Motivation	7
3. Focus of the project	8
4. Requirements	9
5. Contribution	9
Chapter II RELATED WORK	
1. Reflective marker based	11
2. Passive marker and feature based	11
3. Color and histogram based	13
Chapter III EDGE-BASED BLOB DETECTION	15
1. Principles	15
2. Edge detection	16
3. Contour extraction	17
4. Contour classification	19
Chapter IV BLOB DETECTION BY THRESHOLDING	22
Chapter V BLOB DETECTION USING ACTIVE BLOBS	25
1. Principles	25
2. Implementation and related issues	27
Chapter VI BLOB DETECTION BY COLOR SEGMENTATION	30
1. Introduction and principles	30
2. Hardware	30
3. Color representation, HSV conversion and segmentation	30
Chapter VII BLOB DETECTION BY USING CAMSHIFT	34
1. Introduction and principles	34
2. The CamShift algorithm	34
3. Multi-blob detection with CamShift	36
Chapter VIII RESULTS AND DISCUSSION	37
1. Results	37
2. Discussion	42
3. Future development	44
4. Conclusion	45
References	46

# CHAPTER I

## INTRODUCTION

Image understanding and object tracking have long been an important branch of computer research and is a challenging and important problem in computer vision in particular. They involve a wide range of applications, such as security and surveillance, virtual/augmented reality (VR/AR), user interfaces, driver assistance and video abstraction. While including many processes, the critical part of tracking is object recognition. Generally speaking, object recognition is to recognize the model of interest from the input images.

With regard to perceptual user interfaces, tracking systems in these interfaces are usually based on electro-magnetic, optical, inertial, acoustical, and mechanical sensing approaches. Among these, optical tracking has drawn more attention from computer researchers in the recent years due to its low costs, simple implementation and deployment, and portability [1] [4] [5]. In optical tracking systems, tracking tasks often involve recognition of pre-designed markers [1] [5] which are basically patches of a certain shape and color and are attached on the user input devices (probs). Once the locations of the markers are found from the input images, the object can be reconstructed. Used in virtual reality, this allows for natural interaction between human and computer.

Adopting the similar paradigm, the Personal Space Station (PSS), a VR/AR visualizing system at the Dutch Center for Mathematics and Computer Science (CWI), is using marker-based tracking methods for retrieval of user input information. User input devices are marked with patterns made from circular blobs. Recognition and reconstruction of the blobs (markers) in the computer allow the tracking software to know the pose of the input devices. By coupling the input device to a virtual object, this allows the user to have natural and direct access to the object in the virtual world.

In this thesis, I develop and investigate different techniques for real-time blob recognition by using low-end hardware components. The thesis is organized as follows: chapter I introduces the problem, the PSS, the motivation, focus and contribution of the research; chapter II discusses related work; chapter III covers the “Edge-based blob detection” techniques in detail; chapter IV represents “Blob detection by thresholding” , an approach which is, in some sense, a variation of “Edge-based blob detection”; chapter V covers a novel method proposed,

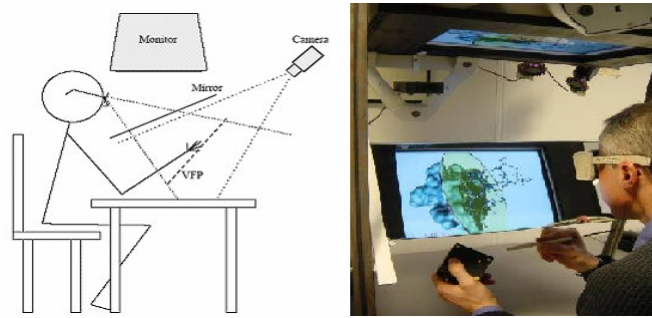
“Blob detection using active blobs”; chapter VI and chapter VII represent two methods which make use of color information for blob recognition, namely “Blob detection by color segmentation” and “Blob detection by CamShift”; chapter VIII provides an in-depth look at the results and discussion besides drawing the conclusion from the research. Also in chapter VIII, comparison of the performance of the proposed techniques will be represented, and trade-off, pros and cons of each technique will be identified in order to make a decision on which method is most applicable for the PSS.

## **1. The Personal Space Station**

CAVE-like systems for VR/AR applications are often extremely expensive and complex in development and deployment. Those systems, thus, provide limited access to their use. Even though CAVE-like systems provide a fully immersive environment, this is not essential in many applications. Over the past several years, there has been a great interest in building more portable and affordable VR/AR systems. Known as *fish-tank* or *desktop* systems, they are able to facilitate a wide range of applications where full immersion is not required.

The Personal Space Station (PSS) is an affordable desktop environment for near-field VR/AR and has been developed at the Dutch Center for Mathematics and Computer Science (CWI) since 2000. While CAVE-like systems are far from being portable and affordable, the main objectives of the PSS are: to provide an environment for 3D applications based on wireless, direct and natural interaction; to provide an environment that can be used under normal office lighting conditions; to allow a low-cost system to be built [2]. To achieve these goals, the PSS uses optical tracking for user’s interaction, thus it is wireless and the user is able to have natural access to the virtual world. Besides, the PSS is built out of off-the-shelf commodity hardware components to keep the costs of the system as low as they can be.

In principle, the PSS consists of a half-reflective mirror in which a stereoscopic image of the virtual world from the CRT is reflected. The objects in the virtual world will then appear to the user as if they came from the VFP under the mirror. The user is comfortably seated in front of the mirror with his elbows rested, reaching under the mirror to interact with the virtual objects directly with his hands or by using graspable, task-specific input devices [Figure 1].



**Figure 1. The prototype PSS. Left: concept, Right: the PSS in practice**

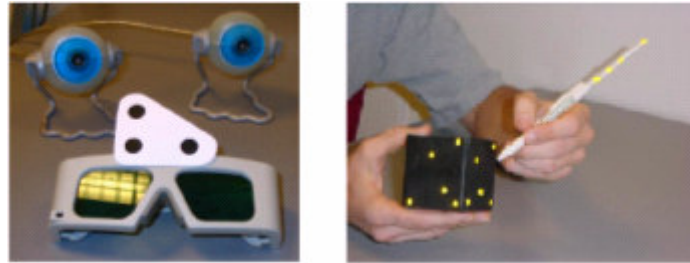
In terms of design, the principle differences between the PSS and other near field VR/AR systems are that in the PSS, the user is equipped with a head tracking system and all 3D interactive tasks are based on optical tracking. The head tracking system allows the user to always have a correct perspective view of the objects in experiment. The advantages of optical tracking are that it is wireless and is less susceptible to noise (e.g. noise produced by the CRT monitor or any surrounding electronic devices).

The tracking engine of the PSS includes two parts: tracking of user input devices and tracking of the user's head. Currently, two separate techniques are being used for each of the trackers. Tracking in interaction space uses infrared camera in companion with retro-reflective markers while the head tracking system makes use of off-the-shelf cameras together with simply painted dot patterns.

For input object tracking, two Leutron Vision LV-7500 progressive scan CCD cameras are used to track the volume in which interaction takes place. Input devices are attached with retro-reflective markers which are used to reflect IR light. The tracking space is illuminated by rings of IR LEDs mounted closely around the camera lenses and IR-pass filters in front of the camera lenses allow only light in a certain range of wavelength to pass through. Therefore, IR light from the markers reflects to the camera lenses and will result in white blobs in the acquired image after a simple thresholding process. The coordinates of these white blobs will be extracted by a blob detector.

The head tracking system uses two FireWire iBOT cameras mounted to the PSS's chassis in front of the user. A simple circular dot pattern is attached to the shutter glasses which are worn by the user. The cameras track the dots and the inter-dot distances can be used to identify the pose of the shutter glasses. The head tracking system is known to be working at 30 updates per second and produces a low computational cost [1].

In the two trackers, once the locations of the markers have been found from the input image frame, the software can easily reconstruct their poses in real world by using the geometric properties of the devices and the configurations of the patterns.



**Figure 2. Left: The head tracking hardware: 2 iBOT cameras and the shutter glasses with the dot pattern attached, Right: retro-reflective markers are pasted on the cube device to be tracked**

The PSS has been (and/or is going to be) installed at various Dutch institutes, some of which are the Department of Biomedical Engineering - Eindhoven University of Technology, Department of Industrial Design - Eindhoven University of Technology and the University of Amsterdam - Section Computational Science.

## **2. Motivation**

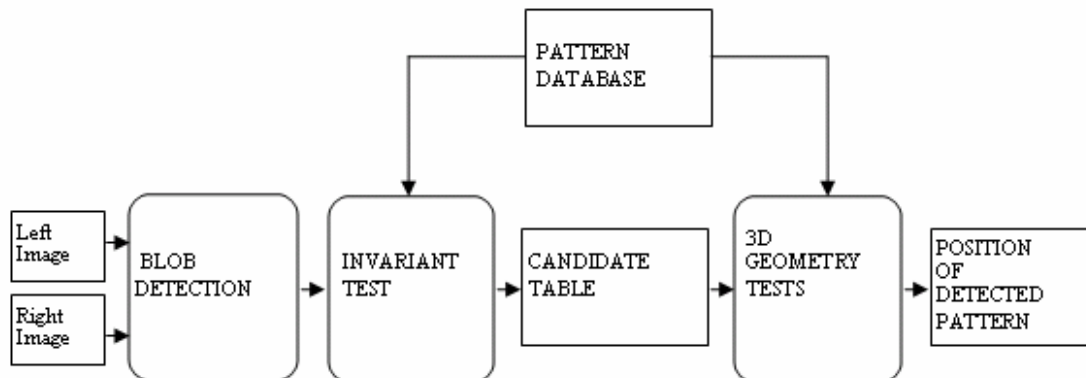
While one of the major motivations of building the PSS is to create an affordable VR/AR environment, lowering the costs of the system is of the utmost importance. At the current stage, the hardware of the prototype PSS costs approximately 13k Euros; more than half of which has been spent on the optical tracking hardware or the user input device tracker to be more specific. Making the costs of the entire system as low as possible will enable more access to this personal VR/AR visualizing system and thus, increase the possibility of bringing it into real-life applications. As a matter of fact, the interaction tracking system using infrared cameras are working nicely, providing accurate results with low latency. The technique has also been used in many other VR/AR systems due to its robust performance [4] [14] [25]. The only problem with the current system is that, the Leutron Vision Pictport H4D dual channel frame grabber and the Leutron Vision LV-7500 progressive scan CCD cameras do contribute a fairly large amount to the final costs of the station. A natural solution for this problem could be using off-the-shelf commodity hardware while retaining (or improving) the system's performance. A possible approach would be using interlaced CCD cameras to reduce the costs of the system significantly.

Furthermore, the existing head tracking system using two FireWire iBOT cameras is reportedly known to have some limitations [1]. As in [1], applying the current head tracking system for interaction tracking is not advisable. From informal discussions with Jurriaan D. Mulder and Arjen van Rhijn, the system is known to be sensitive to changes in illuminating conditions and background noise. For that reason, when working in a normal office working condition where changes in lighting may be expected, the tracker would need to be able to adapt to these changes.

A new approach for robust optical tracking using commodity interlaced CCD cameras for the two trackers of the PSS would be sufficient for the problems addressed.

### 3. Focus of the project

As in the current tracking systems, once blobs have been extracted from the input image, they will be put into a pool of candidate blobs. Next, the pose of the object will be reconstructed using this pool of candidates along with the geometric properties of the user device (or the head mounted pattern in the case of the head tracker) and the formation in which blobs are arranged. Algorithms for object reconstruction are available and well-known in literature; and the 3D reconstruction algorithm used in the current PSS is working very well so that there is no need for improvement. It has been shown to be extremely robust and accurate [5].



**Figure 3. Blob detection in the current user device tracking system**

In this project, I concentrate on developing different methods for recognition of blobs which are attached on the input devices from images grabbed by low-cost interlaced CCD cameras. Once this has been done, the outcome can be applied to the existing 3D reconstruction software without much modification or adaptation. Figure 3 shows the position of blob detection in the user device tracking system of the PSS.

## 4. Requirements

As an experimental session in a VR/AR system usually involves a lot of interaction in real time, any tracking system to be used in a VR/AR system needs to be robust and also works in real time. Furthermore, a successful method needs to be able to work in various lighting conditions. As 3D reconstruction does not produce a big computational load (given an accurate set of candidate blobs), the performance of the entire tracking system will be very much dependent on the performance of the blob detector. In order to be used in the PSS, a blob detector (recognizer) should meet the following requirements:

**Run at the rate of image captured.** For a VR/AR system, the trackers should be as fast as possible. Tracking deficiencies will cause unrealistic behaviors of objects (e.g. leapfrog instead of smooth movement). In the ideal case, the tracker should be able to run at the update rate of the cameras.

**Robust to various lighting conditions.** Lighting condition is a very important factor in computer vision applications in general and tracking systems in particular. Changes in lighting conditions may degrade the performance of the system significantly, or in the worse case, the system may not even work at all. In order for a tracker to work in normal office lighting conditions, it will have to be robust in a wide range of lighting conditions and be able to retain its performance when there are any sudden changes in illumination (time-varying illumination). In chapter VIII, behaviors of the proposed methods against different lighting conditions will be discussed in detail.

## 5. Contribution

In this research, five different blob detection methods have been built. In order to judge the pros and cons of each method, a thorough comparison for the performance of each method is provided. As will be mentioned again in Chapter II, there has not been much work in the area of blob detection. The recognition methods proposed in this thesis are all unique and have not been used in any other tracking systems. These methods have been built from a careful study of the problem and the available and viable techniques to be applied. To be more specific, different well-known algorithms in literature have been studied and chosen to be implemented in different components of the recognizers.

The thesis also serves as a formal comparison for different possible approaches for blob recognition. Based on these results, a specific method may be chosen for deployment or for further improvement due to its performance over the others.

Furthermore, at the best of my knowledge, the “Blob detection using active blobs” method proposed here is the first of its kind. Despite not being as efficient as some other methods at the current time, it does have some advantages over the others. As will be discussed in chapter VIII, with some further improvement, it may be the best choice for some specific applications.

## **CHAPTER II**

### **RELATED WORK**

As a matter of fact, most of the available optical tracking systems at the current time use retro-reflective markers for the tracking tasks and tracking by recognizing patterns from the input images captured by normal CCD cameras is relatively new. With regard to object tracking in general, most algorithms found in literature involve tracking of eyes, faces, pupils, cars and so on. Many of them have been found to be of little use for the research. Furthermore, the rigorous requirements of the project that the blob recognition algorithm must be robust and be able to work in real-time have filtered out a good amount of those techniques. During my research, the following are found to have some connection to the research being carried out.

#### **1. Reflective marker based**

Most of the optical tracking systems [4] [5] [14] [25] are now using IR light in companion with retro-reflective markers for object tracking. The advantage of this is, by choosing the markers appropriately, they can be found easily in the images acquired by the cameras. As the IR-pass filters in front of the camera lenses only allow infrared light of a certain wave length to pass through, the images obtained are relatively “clean” with the markers represented in light regions in a dark background. Blobs in those images can be extracted by using a simple thresholding procedure. In images taken from interlaced CCD cameras, the images are usually much more complicated with other objects presenting in the scene together with the device itself and the background. Extracting blobs from those images would require more complicated and dedicated algorithms.

#### **2. Passive marker and feature based**

Using circular patterns as blobs, it would suggest that a good ellipse detection algorithm would be relevant to the problem at hand. Indeed, in literature, there have been many algorithms proposed to solve the problem.

The Hough transform is one of the best-known algorithms for shape recognition in general and ellipse recognition in specific. In case of ellipse, the Hough transform requires a 5-parameter space which represents the ellipse. Even though accurate, at a glance, the Hough

transform can not be used in a real time tracking system due to the huge computational work load it produces.

Kim et al. [15] observe that, ellipses in an edge image comprise short straight lines. These short straight lines in turn are made of a number of line segments. The algorithm first finds all the line segments in the edge image and merges them to obtain a collection of lines. Next, based on the tangents and neighborhood relations between lines in the line set, appropriate subsets of lines are merged into ellipse arcs. Finally, these ellipse arcs are once again merged into an ellipse by using least squares ellipse fitting. As in the results introduced by the authors, the method can extract ellipses with high accuracy, robustness and fast speed. However, for use in a real time application, the proposed method is still not sufficient. As shown in experiment, in the best case, the method reconstructs an ellipse in 0.37s. This may be fast enough for use in many other applications but certainly not in a real-time tracking system.

Besides the above methods, other well-known methods for ellipse recognition include [16] [17] [18]. Likewise, these methods have been shown to be able to detect ellipses robustly and accurately. The problem is that, they are either built under the assumption that the input images have been processed in some way and only contains the well-defined ellipses or they are simply not fast enough for use in our system.

In the current optical head tracking system of the PSS, the algorithm used by Mulder et al. searches for candidate blobs in the images and put them in to a pool of candidates. The 3D pose of the user's head will be determined by reconstructing the blob pattern in the computer. In order to search for candidates, three steps are performed: edge detection, flood fill and ellipsoid check. The steps performed will make use of information of the blobs found in previous frames, such as, blob positions, blob size. The edge detector works by scanning through the images and tracks any transitions from dark to light and vice versa. If a light-to-dark transition is followed by a dark-to-light transition, plus that the number of consecutive dark pixels is corresponding to that of the current blob size, there should have been a candidate blob. If it is confirmed that a similar behavior exists in the neighbor scan lines, the region is considered a candidate. In order to proceed further, all the dark pixels in the candidate region are checked whether they have the similar gray level. For each region, a local threshold flood fill procedure is performed with the threshold is the darkest pixel of the region. Next, the candidates will go through an ellipsoid check in order to be considered resulting blobs. Since all the blobs attached in the head mounted pattern are circular, all the

resulting blobs detected from the images should have ellipsoid characteristics. The head tracker using this algorithm is robust; it has a hit ratio of 100% and works at 30 frames per second as represented in the experimental result. The major disadvantage of the system is that, it is sensitive to changes in lighting condition and background noise. The tracker would fail in a poorly illuminated environment or where there is too much noise in the images. Besides, in case of head tracking, in general, movements of the blobs are much simpler compared to those of blobs in user device tracking. A general blob detection method should be able to detect blobs in a wider range of camera angles.

### **3. Color and histogram based**

In years, it has been shown that, color information of an image does play an important role in object recognition and object tracking. The inherent problem is, color-based methods are usually extremely sensitive to noise caused by changes in illumination.

Generally, color information of an image may be used in two ways. The color histograms of an image can be manipulated in such a way that, the color information they contain which represent an object will be used for tracking. Generally speaking, in this family of methods, an object of interest which has a certain color will be treated as a color distribution in the input image. The algorithms will then locate this distribution in the image as well as extract all its characteristics (size, bounding rectangle, orientation etc). Alternatively, colors in an image can be used for feature segmentation. Features of the input image can be extracted by color segmentation and go through other processes for object detection.

A color-based tracking algorithm which has been widely used is known as Continuously Adaptive Mean Shift (CamShift) [6]. Besides its pros and cons which will be discussed later in chapter VII, the algorithm is only able to track a single object at one time. In the “Blob recognition using CamShift” discussed in chapter VII, the CamShift algorithm will be used as a component of the blob detector.

Xu et al. [13] proposed a general method for tracking of non-rigid objects. In this method, the user is asked to select a region in the image for tracking. The selection region is down-sampled using Gaussian pyramid sub sampling to smooth the image. Color of the region will then be segmented using Multiple Color Thresholding. For noise reduction, a Color Cluster Window Filter is applied for each pixel. To prevent a big region to be fragmented in to smaller ones, run length coding is used for region grouping. Next, there will be a background

subtraction process such that only the foreground will be extracted. In order to recognize non-rigid objects, external edges of the object are extracted and are refined by minimizing the snake energy of each of the returned contours in the binary image. There are several things in this method which are not relevant to the project at hand. Firstly, in this method, the user needs to specify an initial tracking region to start tracking. Secondly, foreground extraction may not help in the optical tracking systems in the PSS because in these cases, the foreground (the moving part) of an image tends to occupy the major part of the image. Thirdly, the method is for tracking of a single object, and in our case, we need to be able to track multiple blobs which are coming in to, and going out of, the scene in time.

There exist algorithms for color-based multi-object tracking [19]. The color-based tracking algorithm proposed by Pylkko et al. does allow multiple object tracking. The tracker is designed to work with the Nomad XR4000 robot platform in which the robot interface with the visual tracking system via object markers (specifications about the object to be tracked in the scene). The algorithm first reads in the input image and labels all the pixels of the image by utilizing color judges. Basically, labeling indicates that which pixel is of the color of interest. The color judges are actually made of color segmentation algorithms. They can be used concurrently in order to give the possibility of on-the-fly switching between algorithms. Next, connected components from the labeled pixels are extracted and are treated as object candidates. Noise removal is done by eliminating candidates which do not satisfy a certain criterion depending on the geometry properties of the object being tracked.

Generally, the latter two methods use the similar paradigm: color information of the object is used in conjunction with its external edges for tracking. As this is the generally used scheme for color tracking, a similar approach is relevant for our blob tracking problem. Using a similar approach, in chapter VI, I develop a method which combines color properties together with edge information of the blobs for recognition.

## CHAPTER III

### EDGE-BASED BLOB DETECTION

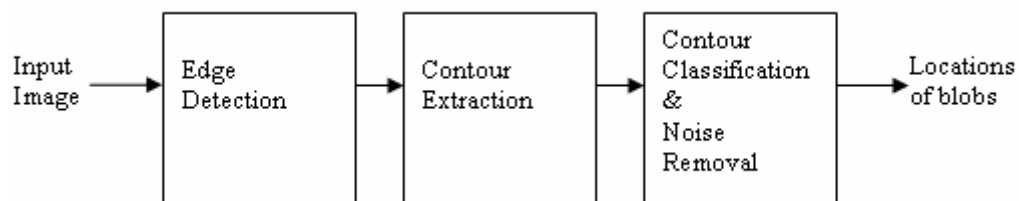
#### 1. Principles

The method utilizes monochrome input images of the device to be tracked. In order to obtain highest contrast between the blobs and their background, the device is attached with black dots on a white background [Figure 4]. The known arrangement of the dots in each facet of the device will be used by the 3D reconstruction algorithm in reconstructing the pose of the device in a later process. The camera used in experiments is the commodity FireWire iBOT webcam which is currently being used for the head tracking system of the PSS. The camera is capable of taking pictures at 30 frames per second and allows adjustment of camera parameters by software.



**Figure 4. Black dot patterns attached on the device painted in white**

The basic principles of the edge-based blob detector are as follows:



**Figure 5. System view of edge-based blob detector**

In this method, each blob in the image is recognized based on the contour surrounding it. The algorithm consists of three steps as depicted in Figure 5. After a frame has been grabbed from the camera, the method first acquires an edge map of this frame by an edge detection procedure. Next, a contour detector will extract all the contours from the edge image. As our blobs are ellipsoid, in order to eliminate a large amount of noise in the edge image, only closing contours will be extracted. Each contour can then be considered a candidate blob. As

the geometric properties of the blobs are known in advance, the pool of candidate blobs will then go through the classifier in order to get rid of all the false candidates. The positions of the remained candidates are then reported as blobs found from the input image.

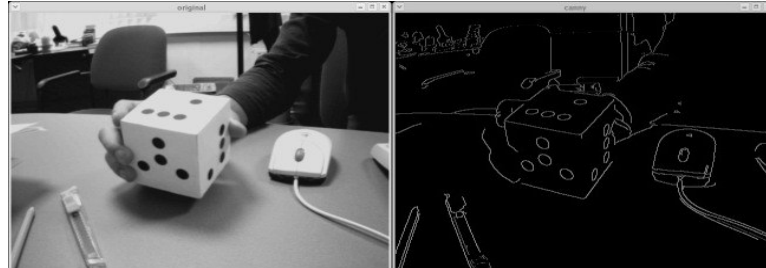
## **2. Edge detection**

Many well-known algorithms for edge detection exist in literature [22]. This method makes use of the Canny edge detector [20], a widely used edge detection scheme currently and is known to many as the optimal edge detector. Canny edge detector takes as input a gray-scale image and produces an image with the positions of gradient changes. To guarantee a low error rate, the detector first smoothes the image by Gaussian convolution to eliminate noise. Then a simple 2-D first derivative operator is applied to the smoothed image to highlight regions of the image with high spatial derivatives. Edges are ridges in the gradient magnitude image. In order to produce thin edges as output, the algorithm then tracks along the top of these ridges and suppresses all pixels that are not at the maximum. The process is also known as non-maximum suppression. The gradient image will then go through a hysteresis process for further reduction. Hysteresis uses two thresholds  $T_1$  and  $T_2$  with  $T_1 > T_2$  and tracks along the remained pixels which have not been suppressed. If the magnitude of a pixel is below  $T_2$  it will be deleted from the image (non-edge). If the magnitude is above  $T_1$  it becomes an edge. In case the magnitude is between  $T_1$  and  $T_2$ , the pixel will be deleted unless there exists a path from this pixel to a pixel with a gradient higher than  $T_1$ . This hysteresis makes sure that a noisy edge will not be broken into multiple edge fragments. This is essential and will show to be helpful for the blob detector because we see that, since we are only interested in closing edges, blob detection will fail even if only one pixel of a blob's edge is missing, making it an open edge. Besides  $T_1$  and  $T_2$ , there is a third parameter which decides the degree of smoothing the image will receive from the algorithm. Smoothing makes Canny edge detector different from other methods like Sobel edge detector, where the detectors are more sensitive to noise in the input image.

Apart from its accuracy and quality, one reason why Canny edge detector is chosen to be implemented as the edge detector for blob detection is its speed. Canny is fast enough for use in real-time applications.

It is seen that, the performance of the method against different lighting conditions depends strongly on the performance of the edge detector. Edges defining blobs in the images should be obtained in different experimental illuminating conditions. For that, in the ideal case, the

three parameters of the Canny detector are chosen in such a way that, they eliminate unwanted edges while retaining blobs' edges. Furthermore, several parameters of the camera such as brightness, gain and shutter speed can be adjusted via software in order to create sharp and clear images with well-defined edges and minimal motion blur for the edge detector.

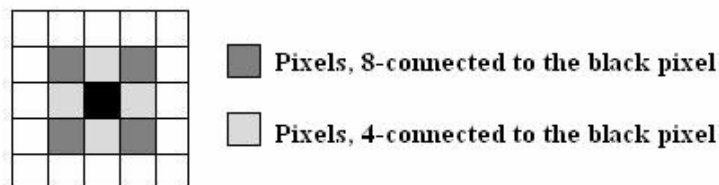


**Figure 6. The input image (left) and the corresponding edge image**

### 3. Contour extraction

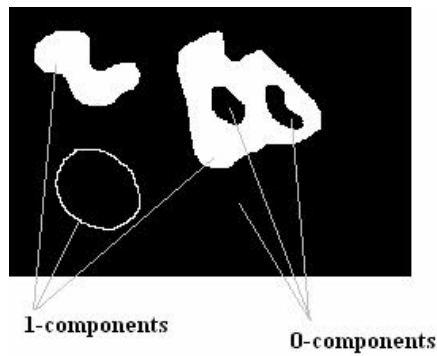
Contours are connected pixels along the border of a region in a binary image. After the edge image has been acquired from the edge detector, contours are extracted. The regions bounded by the contours will then be used in the latter process in order to classify each candidate blob found.

A binary image contains only 0-pixels, i.e. pixels with the value 0, and 1-pixels, i.e. pixels with the value 1. Between pixels belonging to an image, there are two sorts of connectivities: the 4-connectivity and the 8-connectivity. Two pixels with the coordinates  $(x,y)$  and  $(x',y')$  are 4 connected if  $|x-x'| + |y-y'| = 1$ . The two pixels are 8-connected if  $\max(|x-x'|, |y-y'|) = 1$ .



**Figure 7. 4-connected and 8-connected pixels**

The edge image introduces regions made by 0-pixels and those made by 1-pixels. A 0-component is a region made by 0-pixels; likewise, a 1-component is a region of 1-pixels. 0-components make the background while, as edges detected in the previous step are of width 1 pixel, 1-components are those edges. A border point of a 1-component could be any pixel belongs to the component and has a 4-connected 0-pixel. A set of adjacent border points is called a border, or a contour.

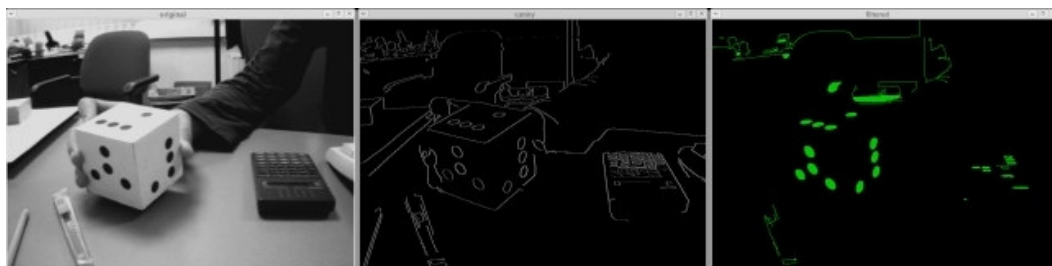


**Figure 8. 0-components and 1-components in the binary image**

Each 1-component has an outer contour which separates it from the surrounding 0-component and may have inner contours which surround any 0-components inside it.

The contour detector used in this method is based on the contour retrieving algorithm proposed by Suzuki-Abe [21] due to its simplicity, robustness and fast speed. The Suzuki-Abe algorithm retrieves contours from the binary image by raster scanning the image to look for border points. Once a point that belongs to a new border is found, a border following procedure is applied to retrieve and store the border in the Freeman chain format. During the border following procedure, visited border points are marked with a special value. The algorithm outputs a list of contours using Freeman chain code.

In order to eliminate noise caused by fragmented edges in the edge image, we only extract regions whose widths are greater than 1 pixel. Furthermore, for regions like the ellipse in Figure 8, there exist two almost identical contours. See that, in this ellipse, the region bounded by the white pixels is an inner region and has its own bounding contour. The 1-region which covers the 0-region also has its own bounding contour. These two contours are just slightly different and actually represent the same object; therefore, only one of them will be extracted. Figure 9 shows the extracted closing contours from the edge image.



**Figure 9. Contours extracted from the input image. Left: input image, Middle: edge image, Right: contours extracted and highlighted by flood-fill**

#### 4. Contour classification

Since our blobs are of circular shape, in 3D space they must have ellipsoid characteristics. In this last step, we do a series of ellipsoid checks in order to throw away any wrong candidates.

Consider the region bounded by a contour a probability distribution function  $f(x,y)$ , the two-dimensional moment of order  $(p+q)$  of this function is defined as:

$$M_{pq} = \iint_{xy} x^p y^q f(x,y) dx dy \text{ or in discrete form: } M_{pq} = \sum_{xy} x^p y^q f(x,y)$$

where  $p, q = 1, 2, 3 \dots$

The definition of the zeroth moment of the function,  $M_{00}$ , represents the total mass of the function; or, in the case of a contour, it represents the total area of the region bounded by the contour.

$$M_{00} = \sum_{xy} f(x,y)$$

The two first order moments:

$$M_{10} = \iint_{xy} xf(x,y) dx dy \text{ and } M_{01} = \iint_{xy} yf(x,y) dx dy$$

or in discrete forms:

$$M_{10} = \sum_{xy} xf(x,y) \text{ and } M_{01} = \sum_{xy} yf(x,y)$$

represent the center of mass of the region bounded by the contour. The coordinates of the center of mass are:

$$\bar{x} = \frac{M_{10}}{M_{00}}, \bar{y} = \frac{M_{01}}{M_{00}}$$

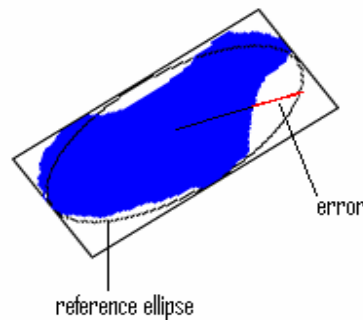
First, for each contour, we calculate the center of mass of the region bounded by the contour. In case of an ellipse this point should coincide with the center of the bounding rectangle of the shape. By this, we eliminate a lot of unsymmetrical shapes.

Further more, there is a threshold set for the ratio between the area of the contour and that of the rectangle bounding the contour.

Using the second order moments ( $M_{20}$ ,  $M_{02}$ , and  $M_{11}$ ), another importance feature of the contour can be determined: orientation. Orientation tells the direction of the principal axis of the contour. The orientation of a contour,  $\theta$ , can be achieved by:

$$\theta = \frac{1}{2} \tan^{-1} \left( \frac{2M_{11}}{M_{20} - M_{02}} \right)$$

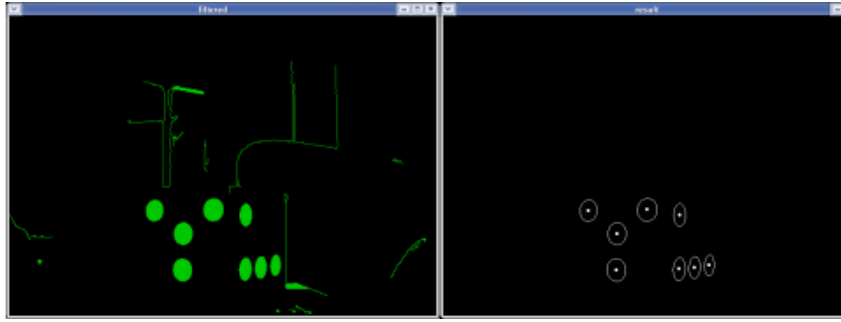
Once orientation is available, it is possible to calculate the minimum rectangle which bounds the contour and has the same orientation. Based on this rectangle, a reference ellipse is determined. This ellipse has the same center as the rectangle, and the length of its major and minor axes are equal to those of the long and short edges of the bounding rectangle.



**Figure 10. The reference ellipse and the error**

Next, using the reference ellipse, for each pixel of the contour an error is calculated. The error is the distance between the pixel and the location at where it is supposed to be. The sum of all the squared errors is taken. A contour is considered qualified as an ellipse when this sum is within a reasonable threshold.

Where ellipses in the images become too small, it is difficult to tell whether they are still ellipses. In that case, even rigorous criteria for classification may fail in distinguishing between an ellipse and another shape (rectangle for instance). For this reason, a threshold for ellipse size is used. In another word, the system will ignore shapes which are too small.



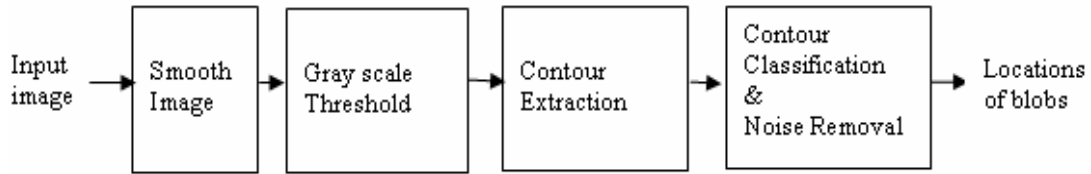
**Figure 11. Blob locations found by the detector**

Figure 11 shows the experimental result. The left image shows the contours extracted from the input image and in the right image are the blobs found. The white dots mark the centers of the blobs.

## CHAPTER IV

### BLOB DETECTION BY THRESHOLDING

In object recognition tasks, object segmentation often plays a critical role in their success. Apart from making use of object edges for segmentation, another possible approach for segmentation is through thresholding. This method processes the gray-scale input images by a thresholding procedure in order to take out the features of interest in the images. The principles of the method are represented in figure 12.



**Figure 12. System view of the method**

In terms of hardware, this method uses the same hardware components as in the previously represented method.

The gray-scale input image grabbed from the camera is first to be smoothed by Gaussian pyramid down sampling [26]. Often, raw images taken from camera contain a lot of noise caused by camera jittering and the sensitivity of the camera's sensor. Therefore, image smoothing is often the first process to be carried out prior to any further tasks. There are many well-known approaches for image smoothing and noise reduction in literature [23] [24].

Gaussian pyramid convolves the input image  $g_0$  with a Gaussian low-pass filter to obtain image  $g_1$ . Thus,  $g_1$  is a reduced version of  $g_0$  in terms of resolution and sample density. Likewise,  $g_2$  can be obtained by low-pass filtering  $g_1$  and so on. Filtering is done by a Gaussian kernel function and the Gaussian pyramid will then be created by the images  $g_0$  to  $g_n$ . Suppose that the input image is represented by the array  $g_0$  comprises  $C$  column and  $R$  rows and each pixel has a value in the range 0 to  $K-1$ . Each pixel of  $g_1$  is computed as a weighted average of value in image  $g_0$  within a  $5 \times 5$  window. The averaging process is done by the function REDUCE.

$$g_0 = \text{INPUT IMAGE}$$
$$g_k = \text{REDUCE}(g_{k-1})$$

For node  $i, j$ ,  $0 \leq i \leq C_1, 0 \leq j \leq R_1$ ,

$$g_1(i, j) = \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) g_0(2i + m, 2j + n)$$

As seen, the width and height of the image  $g_1$  will be reduced by half. The weighting pattern  $w$  called the generating kernel is chosen as:

$$w(m, n) = \hat{w}(m)\hat{w}(n) \text{ and}$$

$$\sum_{m=-2}^2 \hat{w}(m) = 1, \hat{w}(i) = \hat{w}(-i) \text{ for } i = 0, 1, 2$$

To guarantee that all chosen nodes in  $g_0$  contribute equally to nodes in the image  $g_1$   $\hat{w}$  is defined as:

$$\hat{w}(0) = a$$

$$\hat{w}(-1) = \hat{w}(1) = \frac{1}{4}$$

$$\hat{w}(-2) = \hat{w}(2) = \frac{1}{4} - \frac{a}{2}$$

In the implementation,  $a$  has been chosen as 0.4. Furthermore, after the reduced image  $g_1$  is obtained, an image of the input image size will be computed from  $g_1$  by using the EXPAND function. EXPAND interpolates new node values between the given values of  $g_1$ . EXPAND is described as:

$$g_0'(i, j) = 4 \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) g_1\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$$

Image  $g_0'$  with noise reduced will then be used for further tasks. The array represents the amount of noise reduced is:

$$N(x, y) = g_0 - g_0'$$

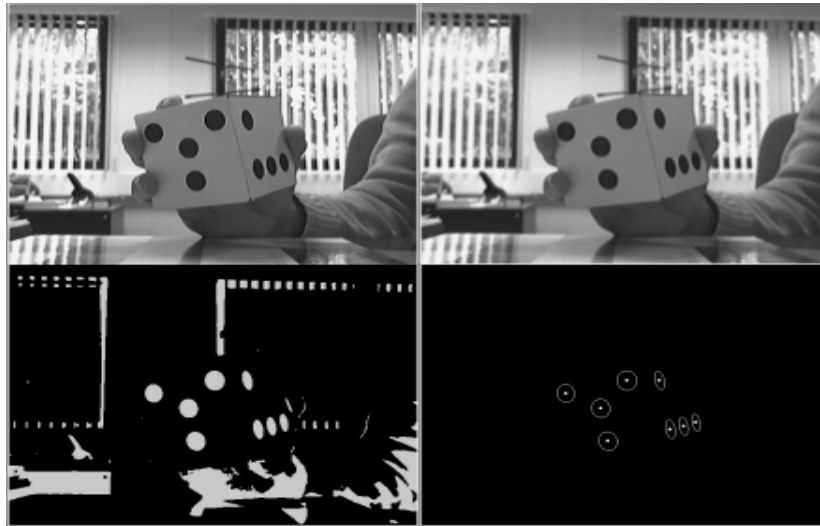


**Figure 13. Noise reduction by Gaussian pyramid. Left: the original image, Middle: the smoothed image, Right: the amount of noise discarded**

Once the input image has been smoothed, it will go through a thresholding process for feature segmentation. Since our blobs are in black, segmentation is done in such a way that it retains all the dark pixels of the image. Pixels with the value below a chosen threshold will result in

white, and pixels with the value above the threshold will result in black in the output gray-scale image.

The image produced by thresholding will contains white regions in a black background. All the connected components in this image will be detected and classified in a way similar to that of the previous method.



**Figure 14. Blobs detected from the input image.**

Figure 14 shows the result. From top to bottom, from left to right: the input image, the smoothed image, the image after thresholding, and the locations of blobs found.

## CHAPTER V

### BLOB DETECTION USING ACTIVE BLOBS

#### 1. Principles

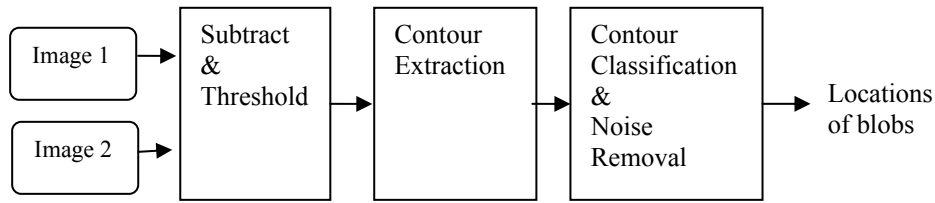
At times, other regions in the image may share the same geometric properties as our blobs'; this may yield false blob detections. Further more, when the working environment becomes too dimly lit, the task of detecting blobs by making use of geometrical features becomes more challenging. That is because, edges, contours and image features, which are often used by geometry-based methods, are usually poorly-defined in images with low level of contrast and are obtained from a bad illuminated environment.

In order to avoid these problems, and yet to pose a novel approach, in this method, the blobs attached to the device are made of Light Emitting Diodes (LEDs) and act as light sources. During tracking, the active blobs will appear to be blinking at a predefined frequency.



**Figure 15. The cube device with active blobs and a close look at the LEDs**

Images are grabbed continuously by the CCD cameras at a speed corresponding to the blinking frequency of the active blobs. Once the cameras and the active blobs are synchronous, it is expected that each continuous pair of images should contain an image with blobs in the “on” stage, and the other image has blobs in the “off” stage. Each pair of images will then be used to determine the current positions of the blobs in those images. By subtracting and thresholding the pair, blobs will appear as white regions in a black background. This resulting binary image will then be used in the classification stage. The principles of the method are depicted in Figure 16. It can be expected that, the tracker built upon this approach can work in very poor illuminating conditions, or, what is more, it is able to track blobs in environments where there is no light at all.



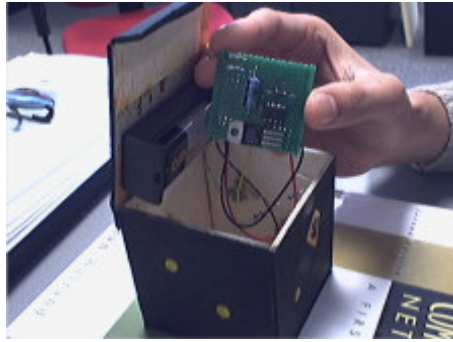
**Figure 16. Principles of active blob detector**

The cube device used in this approach is painted in black. This should help the blob segmentation task since in a fast-moving object tracking session, blobs in the second image may not reside in the same locations as where they were in the first image. In this case, the bright blobs will appear in a black background. Segmenting blobs from the image in this case is trivial.



**Figure 17. A pair of images in the expected case: LEFT: bright blobs, MIDDLE: dark blobs, RIGHT: blobs found after segmentation.**

A circuitry has been developed to allow the active blobs to blink at a desired frequency. The circuitry is a simple one which consists of a 555 timer IC, capacitors and resistors. By adjusting the values of the capacitors/resistors, the frequency of the blobs is determined. In the case of the cameras used, the FireWire iBOT webcams, which are able to take pictures at the speed of 30 frames per second, the blinking frequency of the blobs should be at most 30 Hz (i.e. in a second, blobs will be turned on in 15 times, and off in another 15 times). As a matter of fact, in this approach, the faster the camera is, the more robust and accurate the tracker will be. Since we limit ourselves in using low-cost hardware, off-the-shelf monochrome cameras which are capable of grabbing pictures at 60 frames per second would be most suitable for the task at the current time. However, in order to give a good comparison between all the methods developed during this project, it will be shown later on that, in some extent, the 30 FPS iBOT webcams can still be used to track blobs in real-time.



**Figure 18. The circuitry**

The active cube device is powered by one standard 9V battery and currently weights roughly 70g. This could be a little bulky to use if the technique is to be applied into the head mounting system. In that case, the circuitry could be redesigned to operate using 1 or 2 AA size batteries.

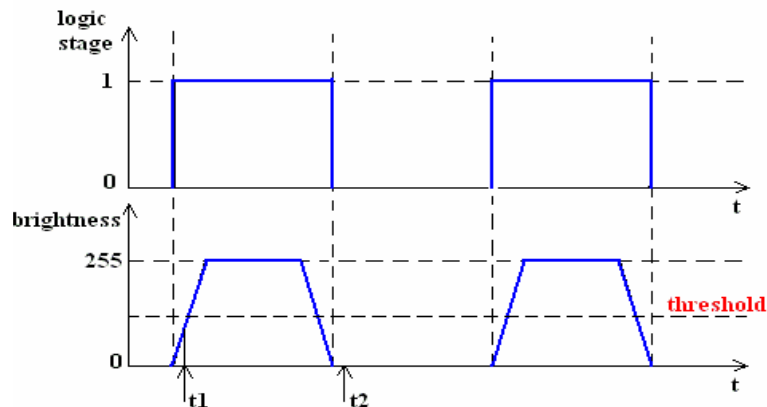
## **2. Implementation and related issues**

In order to build a functional blob detector using this approach, there are several problems to be solved.

First, a critical part of the approach which decides its viability is the synchronization between the camera and the blinking cube device. Since our aim is to build a cordless tracking system, there will be no electric wire to connect the device with the other hardware components. As a consequence, no feedbacks about the current stage of the device will be known directly by the software. The device has been designed in such a way that, the blobs are blinking at a desired frequency. In theory, knowing this value, the camera is easily calibrated so that the whole system is synchronous. The fact is, as commonly known, there are always some errors with the hardware components of the circuitry. However, it can be expected that, the real frequency of the device lies somewhere around the theoretically defined value. Thus, based on this value, a calibrating software program has been written to help find the real frequency of the device. The device is first turned on and positioned in a still background. Camera's picture-taking speed will be adjusted according to the current frequency value of the device. For each frequency within a certain range around the predefined frequency, a series of thousands of images is taken. By checking whether blobs are found in the images after segmentation, a hit ratio for each frequency can be calculated. Finally, the frequency corresponding to the largest hit ratio will be used as the real frequency of the device.

During segmentation, in order to not include slightly different regions in the image pair which are caused by the sensitivity of the camera's lens, a threshold is used. A pixel in the two images is considered changed when the absolute value obtained by subtracting its gray-scale values in the two images must be greater than this threshold value. Certainly, this will prevent us from achieving a hit ratio of 100% even with the correct value of device frequency. In particular, the bigger this threshold is the father the hit ratio will be from 100%. As in experiment, it has been shown that, a hit ratio of 85% can be achieved without tolerating much fault in change tracking.

In Figure 19, the upper chart represents the pulses produced by the circuitry and the lower chart indicates the gray-scale value of the blobs grabbed by the cameras as a function of time. Blobs in images will have their gray-scale values in the range 0-255. As soon as a LED receives a 1 pulse, its stage will change from off to on. However, the change will not be taken immediately. In order to reach the stage of highest brightness, it requires a short amount of time which is generally known as rise time. The figure also shows a typical situation where blobs can not be recognized. Image 1 is taken as time  $t_1$  and image 2 is taken at time  $t_2$ . A blob will fail to be segmented if the absolute difference of its brightness in the two images is smaller than the threshold value.



**Figure 19. LED light signal and change tracking failure.**

The other problem arises when tracking fast-moving blobs. As can be foreseen, using low speed cameras may fail in tracking fast-moving blobs. The reason is, the blobs in the second image may have moved away from their original positions in the first image. Furthermore, since we look for the different regions between the two images, any moving objects in the scene may produce “trails” if the system is equipped with low-speed camera. Practically, moving objects found in a 3D interaction session could be the experimenter himself and other co-experimenters.



**Figure 20. “Trails” produced by the movement of the device**

Quantitatively, in order to archive a frame rate of more than 20 frames per second, which is accounted for the speed of normal movements, the cameras should be able to take pictures at at least 48FPS.

In order to remove unwanted trails, the segmented images will be passed through a classifier. The classifier used here is very much similar to that described in the previous methods. Furthermore, to exclude noise caused by movements in the background, after the first successful frame, the coordinate of the device in the image will be recorded and from the following frame, recognition is only done in the area where the device is supposed to be. If the device moves out of the scene, the process starts all over again.

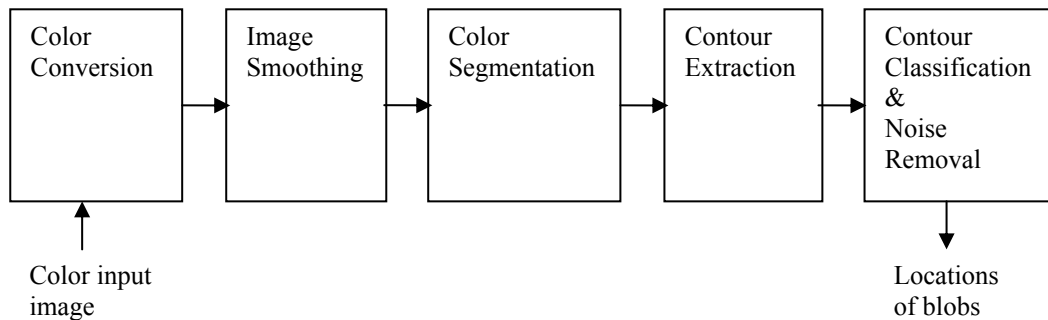
## CHAPTER VI

### BLOB DETECTION BY COLOR SEGMENTATION

#### 1. Introduction and principles

In the previously proposed methods, tracking is done by manipulating monochrome input images taken from the cameras. For years, color information of images has been used for general tracking tasks [6] [9] [11] and this approach has been proved to be computationally cheap, yet the implementation is simple. Generally speaking, these algorithms either extract objects from the scene based on their color properties or, use the image histogram to create a probability distribution function. This function will be used to recognize the objects presented in the images.

This chapter introduces a blob detection algorithm based on color segmentation. A diagram describing this method is represented in figure 22.



**Figure 22. Principles of the blob detector based on color segmentation**

The RGB input color image is first converted to a new color space which is more suitable for segmentation. Then, the image will be smoothed by Gaussian sub sampling as presented in chapter IV. Next, the image will be segmented using the known color properties of the blobs. The extracted candidate blobs will go through the contour detector and classifier in order to throw away any wrong candidates.

#### 2. Hardware

Manipulating color images is a difficult task due to the fact that they are very sensitive to noise. The Firewire iBOT webcam used in the previous methods is very suitable for a real-time tracking system due to its fast data transmission speed. However, during experiment,

color images produced by the camera have been shown to be very sensitive to changes in illuminating conditions. Besides, it also produces a lot of jittering and salt-and-pepper noise as well as wrong color capture. Recognizing objects from those images would be too challenging a task.

A nice substitute which is readily available on the market is the Logitech QuickCam Pro 4000. The webcam connects to the station via USB cable and costs only around €80. What is more, it captures images of high-quality and produces background noise at an acceptable degree. What is best from the Logitech QuickCam Pro webcam is, it is able to work in very low lighting conditions.

Using the same cube device as our widget for testing, colored circular markers are attached on each of its facets. In theory, these markers can be made of any colors. However, in order to avoid color noise produce by the experimenter's skin, colors of red tone should be avoided.

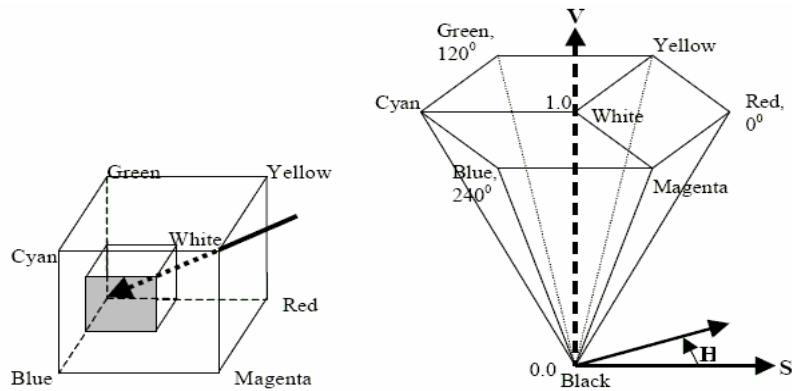


**Figure 23. The device with color blobs and the Logitech QuickCam Pro 4000 webcam**

Among other colors in the spectrum, the green tone appears to have a broader range. For this reason, choosing pure green as blob color would make it more outstanding from other colors in the spectrum.

### **3. Color representation, HSV conversion and segmentation**

Often, color-based recognition methods are very sensitive to changes in illuminating conditions. Color tone of a captured object would change if there were a small change in the direction and the intensity of the light source. While the RGB color system is used widely by most digital images and image capturing devices, it is not suitable for use in recognition tasks. When the value of a separate channel (Red, Green or Blue) changes, the color presentation of the entire image will be affected greatly. For recognition of color objects in varying illuminating conditions, the HSV (Hue Saturation Value) color space is often used instead.



**Figure 24. RGB and HSV color spaces**

HSV in comparison to RGB is shown in Figure 24.

Unlike RGB where colors are mixed up from 3 different color channels, HSV color system separates out color from saturation and brightness. The Hue value represents color while Saturation indicates the intensity of the color and Value contains information about how bright the pixel is. Once an image in the HSV color space is obtained, we will be interested in its colors represented by the Hue channel.

In order to make use of the HSV color space, a conversion from RGB is needed. RGB is converted to HSV using the following scheme:

$$V = \max(R, G, B)$$

$$S = (V - \min(R, G, B)) * 255 / V \quad \text{if } V \neq 0, 0 \text{ otherwise}$$

$$(G - B) * 60 / S, \text{ if } V = R$$

$$H = 180 + (B - R) * 60 / S, \text{ if } V = G$$

$$240 + (R - G) * 60 / S, \text{ if } V = B$$

if  $H < 0$  then  $H = H + 360$

The hue values calculated using the above scheme vary from  $0^\circ$  to  $360^\circ$  so they are divided by 2 to fit into 8-bit destination format.

It is seen from the hex-cone representing the HSV color system that, in very dark places (V value is low) the hue of the image will be very noisy since the hex-cone in this case will be a very small one. The tracker would fail in this case. For that reason, a lower threshold is set for brightness. When saturation (concentration of color) is too low, the pixel simply has no color. Thus, in order to avoid faulty color segmentation, a lower threshold is set for saturation.

The hue value of the green color used for the blobs in this method ranges from 75 to 100. Therefore, these values are set as the lower and upper threshold for segmentation.



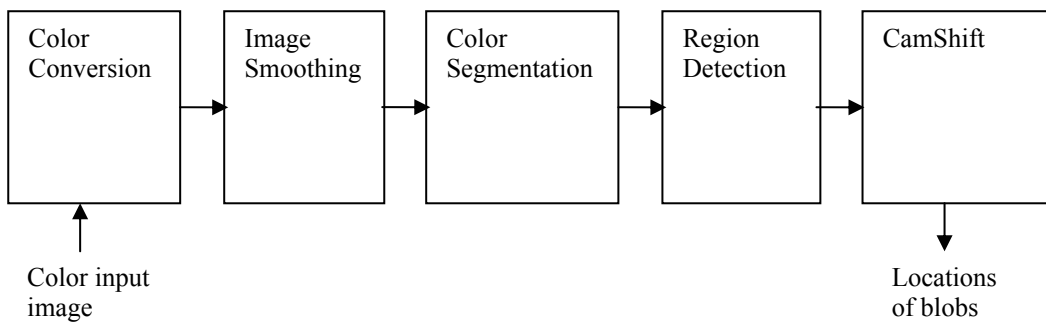
**Figure 25. Experimental result. Left: the original color image, Middle: the image after color segmentation, Right: locations of the blobs detected**

## CHAPTER VII

### BLOB DETECTION BY USING CAMSHIFT

#### 1. Introduction and principles

The CamShift algorithm is a well-known algorithm for tracking of a single color object. In this chapter, a method using CamShift as its core will be proposed in order to adapt the algorithm for our multi-blob recognition problem. The principles of the method are shown in Figure 26.



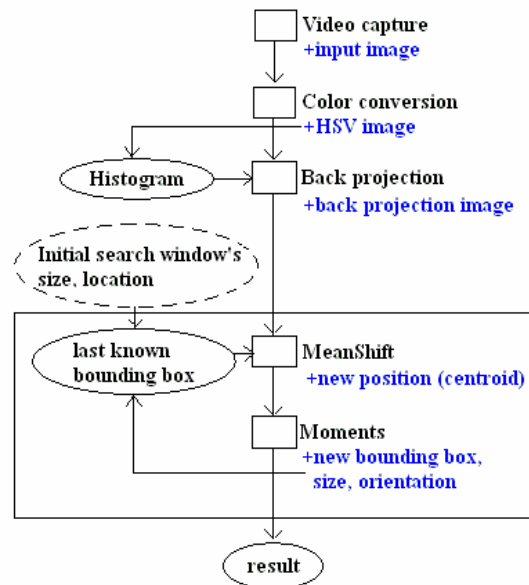
**Figure 26. Principles of CamShift based blob detection**

The first four steps of the methods are the same as those described in the previous method. After the color input image has been segmented, it will be passed to the CamShift algorithm in order to locate the centers of the blobs. The main advantage of using CamShift here is, in case segmentation does not produce high quality ellipsoid regions or the regions have been fragmented, CamShift will treat each region as a probability distribution and locate the center of the region. Thus, it can be expected that in these cases, CamShift for color blob detection is more robust.

#### 2. The CamShift algorithm

The Continuously Adaptive Mean SHIFT (CamShift) is an adaptation of the MeanShift algorithm [12] and was first proposed in [6]. In general, CamShift is an iterative technique to find the mode of a probability distribution. For each frame in a sequence of images, the image is first converted to a color probability distribution function by manipulating the image's histogram. The center, size and orientation of the distribution will be found by CamShift and this information will be passed to the similar process in the next frame. During tracking of a

single frame, the process ends whenever the MeanShift operations converge or an upper bound of iterations has been reached.



**Figure 27. Principles of the CamShift algorithm**

A description of the technique is shown in Figure 27.

The frame captured from the camera is first converted to the HSV color space and the H (hue) channel of the image is thresholded so that only the color of interest is retained. Next, the histogram of the image is calculated and will be used to calculate the color probability distribution image of the desired color which is called the back projection image. Basically, the back projection image replaces each pixel in the frame with the probability that it is a color of interest. Once the back projection image has been obtained, it will be used for the core process of the CamShift algorithm, the MeanShift operator. In the CamShift technique, the user is required to specify an initial window for tracking. Information on the distribution of color probability within the search window is used for the MeanShift operator. MeanShift is fed with the last-known bounding rectangle of the object (or the initial search window in the case of the first frame) and the new position (the centroid) of the region is produced. Using MeanShift, CamShift calculates the center of the local distribution and expand the search window size slightly until convergence. Finally, updated information about the region, i.e. new bounding box, size, orientation, is determined. This is done by calculating the moments of the region and new size, orientation are determine in the similar way as described in the “Edge-based blob detection” method. Updated information about the object found will be used as feedback for the MeanShift operator in the upcoming frame.

The core part of the CamShift algorithm, the MeanShift algorithm can be described briefly as follows:

1. Choose the search window size
2. Choose the initial location of the search window
3. Compute the mean location in the search window
4. Center the window at the mean location obtained in step 3
5. Repeat step 3 and 4 until convergence, i.e. the difference between the old and new location is smaller than the chosen threshold or a certain number of iterations has been reached.

The CamShift algorithm can also be described briefly as follows:

1. Choose the initial location of the search window
2. Apply MeanShift, store the zeroth moment
3. Reset the window size according to the zeroth moment found in step 2
4. Repeat step 2 and 3 until convergence (mean location moves less than a threshold)

### **3. Multi-blob detection with CamShift**

Since CamShift was designed to track a single color object, there are several modifications to use it for our multi-blob detection task. After the segmented image has been available, the Suzuki-Abe algorithm is used again to determine the regions for tracking. For each connected region in the image, the initial search window is set equal to the bounding box of the region. After each blob has been detected, it will be deleted from the back projection image. For that, in case a blob is fragmented into two or more regions, all these small regions will be deleted so that only a single blob is reported.

## CHAPTER VIII

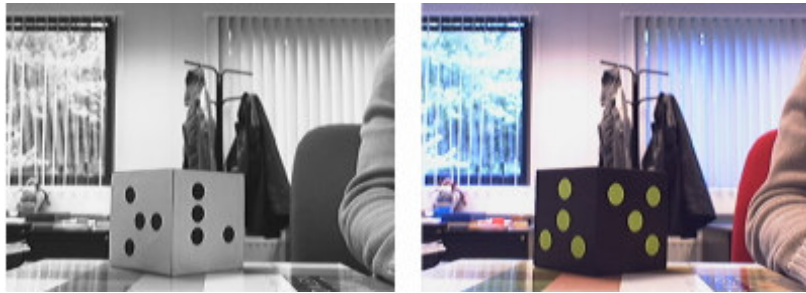
### RESULTS AND DISCUSSION

#### 1. Results

This part describes the performance of the above represented methods. In order to gain a thorough understanding of the pros and cons of each method, comparison for each method's performance against others under the same condition is carried out. The experiments have been performed on an AMD Athlon 2.16 GHz PC running the Fedora Core 2 Linux operating system.

#### Accuracy and speed

In order to measure accuracy and speed of each method, all the experiments are run in a normal office lighting condition. The input dataset is similar to that seen in Figure 28.



**Figure 28. The datasets**

Because different methods detect blobs in different ways, comparing their accuracies is not a simple task. Accuracy of each method is measured as follows: the cube device is placed in front of the camera and blobs in two adjacent facets of the cube which face to the camera will be detected by the blob detector. The cube device is continuously moved from left to right and vice versa. During experiment, moving changes direction when some blobs are about to disappear from the scene (when they have become very narrow). In the ideal case, in each frame taken by the camera, eight blobs should be detected. One thousand image frames are grabbed from the camera and are detected. In theory, the number of detected blobs should be eight thousand. Thus, accuracy is determined by the number of blobs detected divided by eight thousand. For simplicity, the cube device with active blobs has been made with two blobs in each of its facet. Therefore, the number of blobs to be found in each frame will be four. This, however, should not affect the correctness of the measurement. The accuracy

measurement process was carried out for several times, and the average value was taken as the result. The time used to detected blobs in an image frame is defined as the time from when the image has been grabbed until blobs' locations are reported. The results are shown in Table 1. For the color-based methods, the time consumed for each frame is considered the time after the input image has been converted until blobs' locations are found.

<b>Method</b>	<b>Success rate (%)</b>	<b>Speed (ms)</b>
Edge-based	94	2.8
Thresholding	95	1
Active blobs	62	0.96
Color segmentation	97	1.9
CamShift-based	98	2

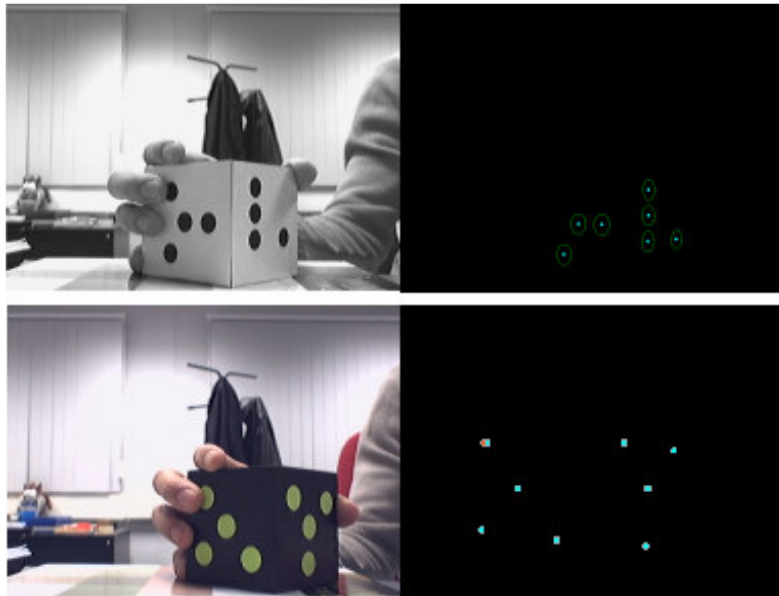
**Table 1. Accuracy and speed of the blob detectors**

As seen from the table, as the principles of Edge-based and Thresholding-based blob detectors are somehow similar, their accuracies are rather comparable in good lighting conditions. In terms of speed, Edge-based detector is almost three times slower than Thresholding-based. This is because the Canny algorithm used for segmentation is much more complex than thresholding. A similar behavior can be found with the Color segmentation and CamShift-based methods. As CamShift is relatively computationally inexpensive, the speeds of the two methods are almost the same. The fastest method, and at the same time, the poorest performer is Active blobs detector. The method is fast because of its light computational load as subtracting images is just as complex as a normal thresholding procedure. One reason is, due to technical limitations, the material used to make the blobs has not been the most suitable one. This makes the ellipsoid regions obtained after segmentation far from being perfect ellipses. In the best case, the accuracy of the method may be expected to be up to 90%.

### **Partial Occlusion**

As for the first 4 methods proposed, blobs are defined as ellipses. When looking for blobs from candidate blobs, the classifier only consider a region a blob if the region has ellipsoid characteristics. Therefore, when a blob is partially occluded, e.g. by the user's finger, the blob detector may not be able to detect this blob. The last method, "Blob recognition by using CamShift" on the other hand, treats candidate blobs as probability distributions and locates

the centroids of these distributions by shifting iteratively. As a result, partial occlusion can be tolerated. Obviously, the position of the detected blob in this case will not be its actual position. However, in some extent, the 3D reconstruction algorithm is still able to reconstruct the pose of the device successfully if the reported blob position is not too far away from the actual position. It may be expected that, the 3D reconstruction algorithm has no problem in finding the pose of the device if one or 2 blobs are half-occluded.



**Figure 29. Detection with partial occlusion. Upper: Edge-based detector. Lower: CamShift-based detector**

Figure 29 shows a case when partial occlusion occurs. The left picture shows the failure of the “Edge-based blob detection” method in detecting the blob partially hidden by the user’s finger. The right picture shows the detected blob center. The red dot seen in the bottom right picture is the actual position of the blob.

### **False detection**

The number of false detections depends very much on several factors, such as the specific input dataset, the speed of the input device and the classifier. In another word, in a less complicated background there will be fewer faults. Since we detect blobs by using their geometric or color characteristics, there might be the case that some other objects of the same characteristics exist in the scene. In this case, it is difficult to avoid false detections. Besides, when blobs become too narrow, it is hard to tell whether they still have ellipsoid characteristics. In order to be able to recognize those blobs, the classifier used in the methods

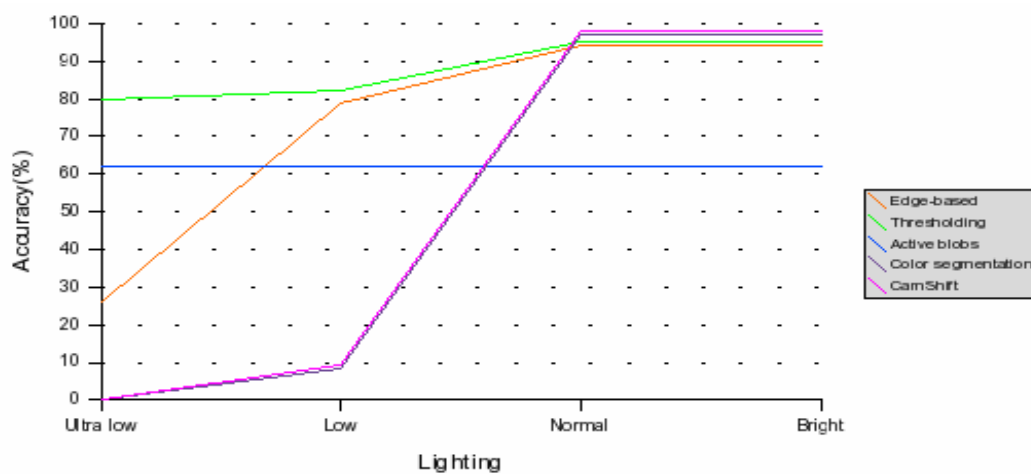
does tolerate some fault in the perfectness of the segmented ellipsoid regions. In these experiments, the input dataset is similar to that described in Figure 28. For the blob detector based on CamShift, it is assumed that no objects of the same color characteristics (light green) should exist in the tracking scene. Table 2 gives a somewhat rough idea of the number of faulty blobs detected by each method using the similar input datasets.

Method	False detections (blob/frame)
Edge-based	0.27
Thresholding	0.65
Active blobs	1.3
Color segmentation	0
CamShift	0

**Table 2. Number of false detections of each method**

### Accuracy against different lighting conditions

To judge the accuracy of the methods against different lighting conditions, the tracking scene is illuminated in different degrees of lighting. Figure 30 shows the behaviors of the blob recognition methods in four degrees of lighting. Normal lighting implies the lighting condition of a normal working office with all the windows closed and lit by ceiling neon lights. Bright lighting is when the office is lit by the ceiling lights and has the windows open to allow ambient sunlight to come in. Low lighting is obtained when half of the ceiling lights are turned off. In this case, only lights in the far end of the office are kept on. Ultra-low lighting indicates the office with all the lights turned off and one desk lamp is placed in the far side of the room.



**Figure 30. Accuracies of the methods in different lighting conditions**

It can be seen that the Edge-based and Thresholding-based methods behave similarly in low to bright lighting conditions. When the environment becomes too dim, accuracy of the Edge-based method falls significantly. Performance of the Thresholding-based method falls slightly to an acceptable level in this case. The reason for this is in low lighting, images taken from the camera become very noisy and Canny fails in extracting the correct edges around the blobs.

As can be predicted, accuracy of the Active blobs method stays the same in all lighting conditions. The method can even work in environments where there is no light at all. In fact, in these conditions, the method is even much more accurate because there is much less noise caused by the movement of the device's edges.

Color-based methods have been shown to be only suitable for working in normal office working conditions. When the scene becomes dimly lit, the blob recognizers are not reliable. This is because in a dark place, there is a change in blob color in the images grabbed.

Furthermore, during experiment, it has been shown that all the investigated methods cope well with sudden changes in illumination.

When tracking a 3D device, if there is an intense light source close to the scene, it can be the case that a facet of the input device reflects light from the light source directly to the camera lens. The facet will then appear in the grabbed image as ultra bright. Sometimes, this may be a problem for detection. To simulate this, a 100 Watts light source is positioned behind the camera and very close to the scene. The cube device is fixed in a position that only one facet is seen from the camera and light from the source reflects by this facet and goes to the camera directly. The accuracy of each method in this case is represented in Table 3.

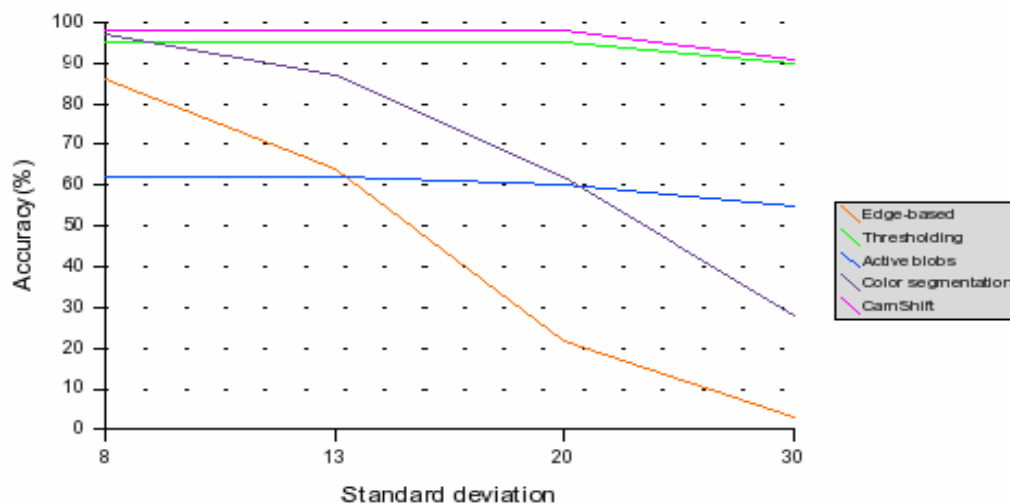
<b>Method</b>	<b>Accuracy (%)</b>
Edge-based	95
Thresholding	0
Active blobs	57
Color segmentation	80
CamShift-based	92

**Table 3. Accuracy of the methods in ultra-bright lighting condition**

The results show that Thresholding-based blob detection fails in the case where a facet of the device receives too much light and this amount of light goes directly to the camera lens. Between the two color-based blob detectors, CamShift-based detector overperforms the method based on color segmentation. The reason is in such a bad lighting condition, blobs tend to be fragmented into several small pieces. For that reason, the classifier of the method based on color segmentation will treat each small region independently and thus, fail in recognizing the entire blob. CamShift, on the other hand, shifts over the whole region and locate its centroid.

### Accuracy against noise

To judge the performance of the methods in noisy environments, different amounts of Gaussian noise is added to the input images for blob detection. Gaussian noise is most frequently seen in digital images and is often caused by the sensitivity of the camera lens or lighting conditions. Therefore, Gaussian noise is evenly distributed across the entire range of frequencies. In this experiment, behaviors of the methods in the present of Gaussian noise of four different standard deviations are studied. The accuracies of the methods can be seen in Figure 31.



**Figure 31. Accuracies of the methods in noisy environments**

## 2. Discussion

As mentioned previously, the speed of each method is defined as the duration of time from after an input image is taken until blobs locations are reported. This, in some way, does not reflect the speed of the tracking systems using the above methods. The frame rates of these

systems also depend on the cameras and the frame grabber interface used and other factors. For example, the color-based blob detection methods proposed require input images in HSV color system. Based on the fact that at the present time, most cameras on the market support raw images in gray-scale, RGB or YUV variations, the two methods first need to convert the input images before any processing tasks. The Active blobs method, on the other hand, requires two image frames for one output frame. This reduces the frame rate of the blob recognizing system significantly. Therefore, among those, blob detection using monochrome input images will be much faster in the current time.

As shown in the experimental results, each blob recognizer has its own pros and cons. Color-based methods are accurate in good lighting conditions but are unreliable in poorly illuminated environments. Both are good in excluding faulty blobs given that there are no objects of similar color characteristics in the scene. Between these two methods, the CamShift-based recognizer performs much better in the presence of noise. Among all, the Active-blob method has been shown to be least robust but it does have some advantages over the others. For example, its accuracy stays the same regardless of how the scene is illuminated. According to the performance of the method, it is not ready to be used in application. The method maybe more applicable when fast monochrome cameras are readily available; cameras with the capability of capturing 60 frames per second, for instance. With some further improvement as will be discussed in the coming part, the method can be good enough for deployment. Of all the five methods proposed, Edge-based and Thresholding-based methods have been shown to be most suitable for use in the current time. They are accurate, and are able to work in a wider range of lighting conditions in comparison to the color-based methods. Between the two, Thresholding-based detector copes better with very low lighting conditions and is much faster. However, in return it fails if some regions of the input image receive too much light and become extra bright. At the moments, these two methods are more suitable for use in a blob tracking system because they work directly with monochrome images taken by the camera. Thus, no image conversion is required and they promise better frame rate. Actually, the two methods have been shown to work at 30 frames per second when using with the FireWire iBOT camera. It is the author's opinion that, for the current time when normal CCD cameras are not really ideal for computer vision tasks, one of these two methods can be used for tracking of blobs in the PSS.

As in many other computer vision systems, the methods may fail when the input device is moving in fast speed and causing motion blur. For this, the camera's shutter speed has been adjusted in a way that it provides sharp images with minimal motion blur. What is more, this

is not too serious a problem since for a normal VR/AR session, it is unrealistic that the input device is moving fast all the time during tracking. In this case, the blob detectors will report the latest blob locations as soon as the device's speed falls in the expected range.

### **3. Future development**

In a sense, the methods proposed in this thesis are not performing the best which they might be expected to. The project, in some way, also serves as a comparison for possible approaches for the blob recognition task. At the current time, the recognition methods can be used immediately with reasonable performances. However, once a favourable method has been selected, it may be advisable to get some improvement in order to get the best performance possible.

#### **Prediction**

It is quite natural that historic information of the previous frames should be used when detecting blob in the frame being processed. For this, recognition will only be carried out on the regions where blobs are predicted to exist. This is quite useful for avoiding faulty detections as well as making detection more efficient. For tracking of the interaction space of the PSS, blob detection using CCD cameras is quite new. Therefore, during the time of this project, the configurations of blob pattern to be used in each facet of the cube device were not known. Once they have been determined, the information on the arrangement of blobs on each facet can be used for prediction. In particular, the Kalman filter can be used to keep track of the locations of the blobs found in the previous frames and to predict their new locations in the current frame. For newly emerged blobs, prediction can be made by using information about the configurations of the blob patterns or by using feedback from the 3D reconstruction module. This improvement can be made for all of the methods above. In particular, it is expected that the performance of the Active-blobs recognizer will be enhanced significantly once this improvement is made.

#### **Adaptive parameter adjustment**

In a given experiment condition, each of the method is more efficient if a right set of parameters is chosen. The main factor which decides whether a set of parameter is the most relevant is lighting. Thus, brightness of the captured input image may be calculated and information on the current lighting condition can be achieved. Parameters can then be

adjusted accordingly. In particular, there are several parameters to be adjusted on the fly. Camera parameters, such as brightness, gain, contrast and shutter speed can directly affect the quality of the pictures. Thus, having a right set of camera parameters is quite essential. Besides, for the methods using Canny, setting Canny's thresholds properly yields better edge images.

#### 4. Conclusion

In this thesis, I have investigated five different methods for real-time blob detection using low-end hardware components. Each method has its own pros and cons. For the time being, it has been shown that the most applicable methods are Edge-based blob detection and Blob detection by thresholding. The methods can be used for the optical tracking systems of the PSS with reasonable performance. Further improvement may be made in order to achieve best performance. The usability of each method in the PSS is described in the table below.

Method \ Lighting	Ultra low	Low	Normal	Bright	Ultra bright
Edge-based	-	+	++	++	++
Thresholding-based	+	+	++	++	-
Active blobs	-	-	-	-	-
Color segmentation	-	-	++	++	+
CamShift-based	-	-	++	++	++

++ Sufficient for use in the PSS

+ Acceptable for use in the PSS

- Insufficient for use in the PSS

## REFERENCES

- [1] Jurriaan D. Mulder, Jack Jansen and Arjen van Rhijn. An Affordable Optical Head Tracking System for Desktop VR/AR Systems – *Eurographics Workshop on Virtual Environment 2003*
- [2] Jurriaan Mulder and Robert van Liere. The Personal Space Station: Bringing Interaction Within Reach – *Richer and Taravel (Eds), Proceedings of the Virtual Reality International Conference VRIC 2002, 73-81, 2002*
- [3] Jean-Bernard Martens, Wen Qi, Dima Aliakseyeu, Arjan Kok and Robert van Liere. Experiencing 3D Interaction in Virtual Reality and Augmented Reality - *ACM International Conference Proceeding Series; Vol. 84, pages 25-28, 2004*
- [4] Miguel Ribo, Alex Pinz and Anton L. Fuhrmann. A new Optical Tracking System for Virtual and Augmented Reality Applications – *In Proceedings of the IEEE Instrumentation and Measurement Technical Conference, pages 1932-1936, 2001*
- [5] Robert van Liere and J.D. Mulder. Optical tracking using projective invariant marker pattern properties. Accepted for Publication in the *Proceedings of the IEEE Virtual Reality 2003 Conference, 2003*
- [6] Gary R. Bradski. Computer Vision Face Tracking For Use in a Perceptual User Interface – *In Intel Technology Journal Q2 '98*
- [7] Michel Boyle. The Effects of Capture Conditions on the CAMSHIFT Face Tracker - *Report 2001-691-14, Department of Computer Science, University of Calgary, Alberta, Canada*
- [8] Alexandre R.J. Francois. CAMSHIFT Tracker Design Experiments with Intel OpenCV and SAI - *IRIS Technical Report IRIS-04-423, University of Southern California, Los Angeles, July 2004*
- [9] Martin Armstrong and Andrew Zisserman. Robust Object Tracking - *In Asian conference on Computer Vision, volume I, 1995*
- [10] Dorin Comaniciu and Peter Meer. Robust Analysis of Feature Spaces: Color Image Segmentation – *In Proceedings of the 1997 IEEE Conference on Computer Vision and Pattern Recognition, pages 750-755, 1997*
- [11] P. Perez, C.Hue, J. Vermaak and M. Gangnet. Color-based Probabilistic Tracking – *A. Heyden et al. (Eds): ECCV 2002, LNCS 2350, pages 661-675, 2002*
- [12] Yizong Cheng – Mean Shift, Mode Seeking, and Clustering - *IEEE. Transactions on Pattern Analysis and Machine Intelligence , Vol. 17, pages 790-799, 1995*
- [13] Richard Y.D. Xu, John G. Allen, Jesse S. Jin. Robust Real-time Tracking of Non-rigid Objects - *In Proceedings of the 2003 Pan-Sydney Area Workshop on Visual Information Processing (VIP2003), Sydney, Australia. CRPIT, 36. Piccardi, M., Hintz, T., He, S., Huang, M. L. and Feng, D. D., Eds., ACS. 95-98.*
- [14] K. Dorfmueller. An optical tracking system for VR/AR-applications. In M. Gervautz, A. Hildebrand, and D. Schmalstieg, editors, *Virtual Environments '99, Proceedings of the Virtual Environments Conference & fifth Eurographics Workshop, pages 33--42, 1999.*

- [15] Euijin KIM, Miki HASEYAMA, Hideo KITAJIMA: "Fast and Robust Ellipse Extraction from Complicated Images," *International Conference on Information Technology & Applications (ICITA 2002)*, pages. 138-5 (2002)
- [16] Markus Vincze, Minu Ayromlou and Michael Zillich. Fast Tracking of Ellipses using Edge-Projected Integration of Cues – *Proceedings of the International Conference on Pattern Recognition (ICPR'00)*
- [17] E.R. Davies. Algorithms for Untra-fast Location of Ellipses in Digital Images – *Image Processing and its Applications, Conference Publication No. 465 IEE 1999*
- [18] Raymond CHAN and Wan-Chi SIU. Fast Detection of Ellipses Using Chord Bisectors - *International Conference on Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990*
- [19] Pylkko, H. Riekkki, J. Roning. Real-time color-based tracking via a marker interface - *IEEE International Conference on Robotics and Automation, 2001. Proceedings 2001 ICRA*
- [20] J. Canny. Computational approach to edge detection - *IEEE-PAMI 8(6):679-698, 1986*
- [21] S. Suzuki, K. Abe. Topological structural analysis of digital binary images by border following – *CVGIP 30(1): 32-46*
- [22] Mike Heath, Sudeep Sarkar, Thomas Sanocki, and Kevin Bowyer. Comparison of Edge Detectors: A Methodology and Initial Study - *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society*
- [23] Rafael C. Gonzalez, Richard E. Woods. *Digital Image Processing, Second Edition. Prentice Hall International*
- [24] Robert M. Haralick, Linda G. Shapiro. *Computer and Robot Vision, Volume I, Addison-Wesley Publishing 1992*
- [25] K. Dorfmüller. Robust tracking for augmented reality using retroreflective markers. *Computer and Graphics, 23(6):795-800, 1999*
- [26] Peter J. Burt, Edward H. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions On Communications, Vol COM-31, No 4, April 1983*
- [27] Intel Open Source Computer Vision Library  
<http://www.intel.com/research/mrl/research/opencv/>
- [28] <http://www.orangemicro.com>