Universit de Caen Basse-Normandie
Master M2 recherche Langage, Image et Document (LID)
April to September 2008

Hand Gesture Analysis and Recognition in Video Sequences

By:
Bahjat Safadi: *bhjat_s@hotmail.com*

Supervisors:

Dr. Youssef Chahir: *youssef.chahir@info.unicaen.fr*
Dr. Michele Molina: *michele.molina@unicaen.fr*

September 10, 2008

Acknowledgements

I would like to thank my thesis supervisor Dr. Youssef Chahir for his sincere efforts to follow all the stages of completing this thesis. I thank Dr. Michele Molina esteemed, who gave a lot of her time for my thesis and who put considerable effort in giving the success for the work requirements. I give my appreciations to the Researchers in the GREYC Laboratory, whose expertise and knowledge have helped me during my thesis. Also, I thank my parents and Maaike for their support and love.

Bahjat

Contents

List of Figures

Chapter 1

INTRODUCTION

Klatzky and Lederman [1] give many evidences that in adults, haptic perception properties are achieved by the execution of specific Exploratory Procedures *"EPs"*. An "EP" is a stereotyped movement pattern which is dictated by the object properties which the haptic system chooses to process, both perceptually and cognitively. Klatzky and Lederman [1] described six basic Exploratory Properties. Lateral motion is an EP aimed at perceiving the texture of an object. It is achieved by rubbing the fingers across a surface. Pressure is an EP associated with the hardness of an object. By pressing down on an object, we gain information about object hardness. This pressure is a force applied on the object while the object is stabilized. Static contact is an EP associated with object temperature. It corresponds to a contact in one spot by a large skin surface without effort to mold to contours. Unsupported holding, which is holding an object away from a support, is an EP associated with object weight. Enclosure is defined as wrapping the hand around an object and provides information about its global shape and volume. Contour following is an EP defined as moving the fingers over the perimeter of an object and provides information about the exact shape of an object. The identification of these exploratory procedures is necessary to design robots but also, for human beings, to interact with machine. Vision based hand gesture recognition methods are generally categorized in two groups: feature based [2] and appearance based methods [3]. In feature based methods, initially, we need to extract model or features from images. In appearance based methods, images can be used directly for hand gesture recognition. Several methods have used motion history images as temporal templates for gesture recognition [4]. Appropriate features are extracted from this image and different methods are used for classification such as Neural Network [5] and the Hidden Markov Model [5]. The aim of the present work is to devise an automatic manual testing procedure able to extract tactile information effectively. The system describes in this thesis, has been conceived to recognize the texture and the hardness of an object through the video analysis of hand actions. This work, based on the seminal description of Lederman and Klatzky [1], is aimed to answer the following question: what are the hands doing?, what is the action

-they are exerted- aimed at? Two objects properties have been tested: texture and consistency. For each property, two modalities were proposed. The texture of the object being explored could either be smooth or granular. Its consistency could either be hard or soft. Consequently, EPs extracted by this system were: Lateral motion: For the smooth object, lateral motion corresponds to a continuous rubbing of the hand all over the object surface. For the granular object, lateral motion corresponds to low amplitude movements of one or more fingers scratching the surface in a left-right or up-down direction. Pressure: For the soft object, the torque exerted on the object induces that the fingers gets closer to some of the others. For the hard object, the torque exerted on the object did not lead to any change in the distance between fingers. We would like to build a system which is able to recognize the action or the behavior of the hands. four EPs are categorized:

1. Lateral Motion for Smooth Object ($LMSO$)
2. Lateral Motion for Granular Object ($LMGO$)
3. Pressure for Soft Object (PSO)
4. Pressure for Hard Object (PHO)

Chapter 2

3D Interactive Object Segmentation

In this chapter we present a technical interactive content segmentation of an image and of a video object segmentation, an approach based on the spread of labels using the graph cut algorithm, which is useful to separate objects from the background in an image. In our project we are analyzing videos to recognize hand gestures. As the first step we need to segment our videos to separate the two hands, *because a gesture can be showed by each hand and also by the two hands together.* By using this technique we succeeded to separate the hands in the videos "left hand and right hand", after this segmentation we can study the movement features for each hand alone.

2.1 Introduction

Image segmentation can be defined as the task of dividing an image into regions that have a strong correlation with objects or areas of the real world contained in the image which means to distinguish objects from background in images. Typically this division is based on low-level cues such as intensity, homogeneity or contours. Four popular approaches based on such cues are:

- **Threshold techniques:** they make decisions based on local pixel information and are effective when the intensity levels of the objects fall squarely outside the range of levels in the background. Because spatial information is ignored, however, blurred region boundaries can create some damage.

- **Edge-based methods:** their weakness in connecting together broken contour lines make them, too, prone to failure in the presence of blurring.

- **A region-based method:** the image is partitioned into connected regions by grouping neighboring pixels of similar intensity levels. Adjacent regions are then merged under some criterion involving perhaps homogeneity or sharpness of region boundaries.

- **A connectivity-preserving relaxation-based segmentation method:** usually referred to as the active contour model, which was proposed recently. The main idea is to start with some initial boundary shapes represented in the form of spline curves, and iteratively modify it by applying various shrink/expansion operations according to some energy function.

The difficulty lies in formulating and including prior knowledge into the segmentation process. How does one describe ones perception of what constitutes foreground in an image through low level cues? As distinguishing between objects and background becomes harder and requires a higher level of scene understanding this task becomes increasingly difficult.
Here we attempt to address this issue, we wish to partition images into two parts. Many approaches were suggested for this issue of interactive segmentation; the approach taken here is based on Graph-cut [**9, 11, 12**] technique. That was motivated by the fact that it is one of the most successful approaches in image segmentation. In addition, it also allowed for a straightforward incorporation of prior knowledge into its formulation. A suggestion for an efficient implementation along with some preliminary results on two different types of images is also given.

2.2 Graph Cuts

A graph cut is the process of partitioning a graph into disjoint sets. The concept of optimality of such cuts is usually introduced by associating energy to each cut. Graph cut methods have been successfully applied to stereo, image restoration, texture synthesis and image segmentation.

2.3 Min-cut/Max-flow cuts

Given a graph $G = \langle V, E, W \rangle$, where V denotes its nodes, E its edges and W the weighting matrix, that associates a weight to each edge in E. A cut on a graph is a partition of V into two subsets A and B such that

$$A \cup B = V, \text{ and } A \cap B = \phi$$

Perhaps the simplest and best known graph cut method is the min-cut formulation. The min-cut of a graph is the cut which partitions the graph G into disjoints segments such that the sum of the weights associated with edges between the different segments is minimized.

$$C_{min}(A, B) = \sum_{u \in A, v \in B} W_{uv}$$

2.4 The Image Seen as a Graph

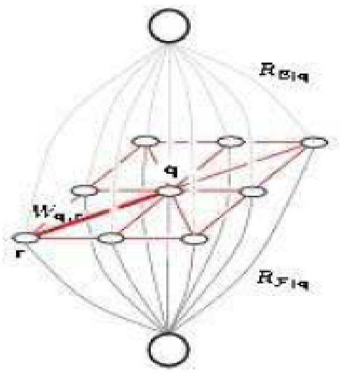


Figure 2.1: Graph representing a 3-by-3 image 2D.

Basically each pixel in the image is viewed as a node in a graph, edges are formed between nodes with weights denotes how alike two pixels are, given some measure of similarity, as well as the distance between them. The edges for each pixel can be formed between the pixel with all the other pixels. In attempt to reduce the number of edges in the graph, we will predetermine neighborhood N that describes the neighbors of each pixel and we will be interested in the similarity "*distance*" between each pixel and its neighbors, Any kind of neighborhood system can be used $(4,\ 8, \cdots)$. There are two additional terminal nodes: an object terminal (a SOURCE) and a background terminal (a SINK).These two terminal nodes do not correspond to any pixel in the image but instead they represent respectively the object and the background. The source is connected by edges to all nodes identified as object seeds and the sink is connected to all background seeds. Edges are formed between the source and sink and all other non-terminal nodes, where the corresponding weights are determined using models for the object and background.

The distance $d_{i,j} = f(|I_i - I_j|)$, where I_i and I_j are the intensities at pixels i and j, and,

$$W_{ij} = K \times e^{\frac{d^2_{ij}}{2\sigma}}$$

Then, the min-cut of the resulting graph will be the segmentation of the image. This segmentation should be a partition such that similar pixels close to each other will belong to the same partition. In addition, as a result of the terminal weights, pixels should also be segmented in such a manner so they end up in the same partition as the terminal node corresponding to the model (object or background) they are most similar to. We compute the edge weights between pixels as the following: the edge weight between pixels i and j will

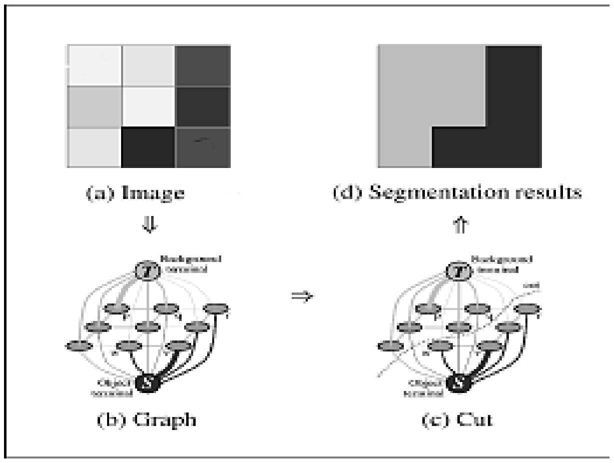


Figure 2.2: Example segmentation of a very simple 3-by-3 image. Edge thickness corresponds to the associated edge weight. (Image courtesy of Yuri Boykov.)

denoted as W^I_{ij}, and the terminal weights (source and sink) between pixel i and j, the source W^S_i and sink W^T_i are given by:

$$W^I_{ij} = K \times e^{-\frac{\|w(i)-w(j)\|^2}{2\sigma}}$$
$$W^S_i = \frac{p(w(i)|i\in s)}{p(w(i)|i\in s)+p(w(i)|i\in t)}$$
$$W^T_i = \frac{p(w(i)|i\in t)}{p(w(i)|i\in s)+p(w(i)|i\in t)}$$

where $r(i;j)$ is denoted to the distance between pixel i and j, K is constant and σ is tuning parameters weighing the importance of the different features. Hence, W^I_{ij},j contains the inter-pixel similarity, that ensures that the segmentation more coherent. W^S_i and W^T_i describe how likely a pixel is to be background and foreground respectively.

2.5 3D Graph

In a video we construct a 3D graph that is obtained from a series of images that describe the video. Each node from the graph is connected to 26 (pixels) neighbors, that means it has 26 edges with weights calculated as described in the 2D graph. We applied the same algorithm as above with some changes such as,

Neighbors=26

Video depth was included

2.6 Implementation

We will detail the new algorithm of Boykov-Kolomogrov*[Boykov-al, 2004]*, which was taken in this thesis as interactive segmentation approach. This algorithm belongs to all the algorithms based on path augmentation, with the construction of two research trees (one from the source and the other from the sink). The two search trees are noted as the consecutive S-tree and T-tree. In the S-tree, the edges of each node parent to its children are not saturated, whereas in the T-tree the edges of the children nodes to their parents are saturated. A node which does not belong to one of these two trees, is called a free node. The internal nodes of a tree are passive nodes; and nodes of the borders are active nodes. The role of the active node is to augment the tree to acquire new children from all the free nodes. When the active node detects a neighbor node from the second tree, it sends a found path augment to the next step. This algorithm repeatedly runs through three steps:

- **Step Growth:** S-tree and T-tree are growing up to find an augmented path.
- **Step Augmentation:** the found path is augmented.
- **Step Adoption:** S-tree and T-tree are restored.

The first step augments one tree (from the two) from a set of active nodes found from the previous iteration. The initial stat which begins with root node is *(S)* or *(T)*, during this step the active nodes explore their unsaturated neighborhood edges and acquire a new free child. This step ends when an active node faces another active node of the tree opposite to describe a root path found.

$$Tree(p) = \begin{cases} S & if\ p \in S \\ T & if\ p \in T \\ 0 & if\ p\ is\ free \end{cases}$$

Parent(p) represents the parent node of p, *Parent(free p)=0.* The notation *tree_cap*$(p \rightarrow q)$ describes the residual capacity of the edge(p,q) if *Tree(p)=S*, otherwise it describes the *edge(q,p)* if *tree(p)=T.*

Function Growth:

```
while $A \neq \phi$ do
  Chose an active node $p \in A$
  for $P_q \in N$   //Neighbors of $p$ do
    if tree_cap($p \rightarrow q > 0$) then
      if ($Tree(q) = \phi$) then
        $Tree(q) = Tree(p)$
        $Parent(q) \leftarrow q$
        $A \leftarrow A \cup q$   //add q as an active node
      end if
    else
      if ($Tree(p) \neq Tree(q)$) then
```

```
            return  $(path = path(s \rightarrow t))$
          end if
        end if
      end for
      $A \leftarrow \{A - \{p\}\}$
    end while
    return  $(path = \phi)$
```

The path p found at the step of growth, must be augmented in the next step of The augmentation with the amount *df* of the possible flow. Some edges (at least one) from the path p must become saturated. The nodes which correspond to the saturated edges from the same tree, are considered as orphan nodes, this can divide the tree into S and T, which are the roots of two trees as orphan nodes from the roots of the other trees.

Function Augmentation:

```
Find df in the augmented path
Modify the residual graph by pushing df on path
for $edge(p,q) \in P$  which becomes saturated do
  if $(Tree(p) = Tree(q) = S)$ then
    $Parent(q) \leftarrow \phi$
    $O \leftarrow \{O \cup q\}$
  end if
  if $(Tree(p) = Tree(q) = T)$ then
    $Parent(p) \leftarrow \phi$
    $O \leftarrow \{O \cup p\}$
  end if
end for
```

The goal of the third step is to restore a single tree root with two raciness S and T, then to find for each orphan a parent in the neighborhood. A parent must belong to the same S or T as the orphan node; also it must be connected to any saturated edge. If there is no parent which satisfies these conditions, the orphan node becomes a free node and its children become orphans. This step ends when all the orphan nodes become empty.

Function Adoption

```
while $O \neq \phi$ do
  Chose un orphan $p \in O$
  $O \leftarrow O - \{p\}$
  Process $p$
end while
```

The function process p seeks a valid parent (in the same tree), where the edge between them is unsaturated. Note that after the adoption, the state of p (passive or active) remains unchangeable. If the orphan node p did not find any valid parent, the process p must execute the following pseudo code:

Process P

```
for $p, q \in N$ do
  if $Tree(p) = Tree(q)$ then
```

```
            **if** $tree\_cap(q \to p) > 0$ **then**
                $A \leftarrow A \cup \{q\}$
            **end if**
            **if** $Parent(q) = p$ **then**
                $O \leftarrow O \cup \{q\}$
                $Parent(q) \leftarrow \phi$
            **end if**
        **end if**
    **end for**
    $Tree(p) \leftarrow \phi$
    $A \leftarrow A \cup \{p\}$
```

2.7 Segmentation results

First, we applied the Graph-cut algorithm on a $2D$ images. Results were good as $2D$ segmentation of the objects in these images, figures (2.3 and 2.4) show results of two images with different σ and λ.

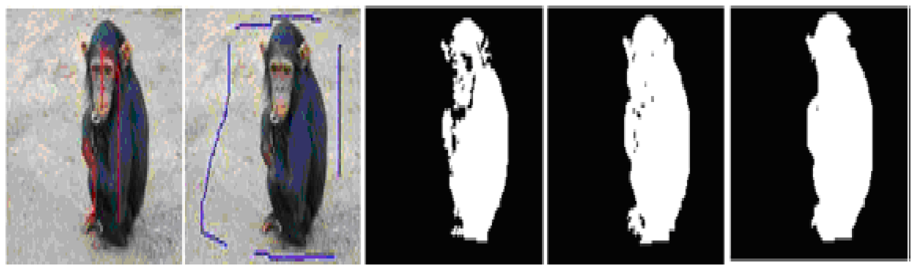


Figure 2.3: 2D image example 1, from left to right, object, background, segmentation($\sigma = 20, \lambda = 0.1$), segmentation($\sigma = 20, \lambda = 0.1$), segmentation($\sigma = 20, \lambda = 0.1$), segmentation($\sigma = 20, \lambda = 0.1$)

Then, it gave quite good results when we applied it on the videos contain two hands. The difficulties of separation the two hands are when the hands are moving on over the other hand, that means when we the two hands are connected in the video. Here it is obvious to select carefully the points where the algorithm can extract well the hand.

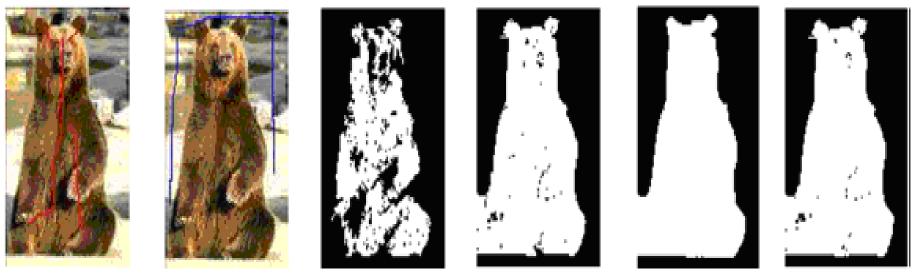


Figure 2.4: 2D image example 2, from left to right, object, background, segmentation($\sigma = 20, \lambda = 0.1$), segmentation($\sigma = 20, \lambda = 0.1$), segmentation($\sigma = 20, \lambda = 0.1$), segmentation($\sigma = 20, \lambda = 0.1$)

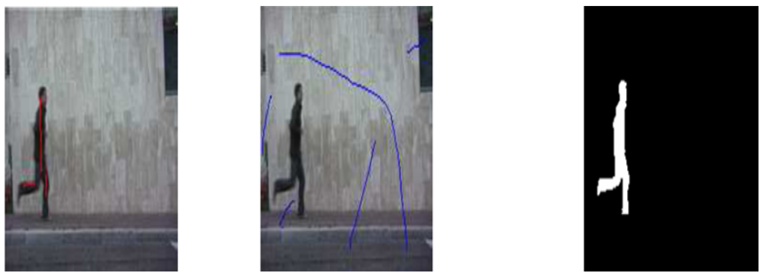


Figure 2.5: 2D image example 3, from left to right, object, background, segmentation result

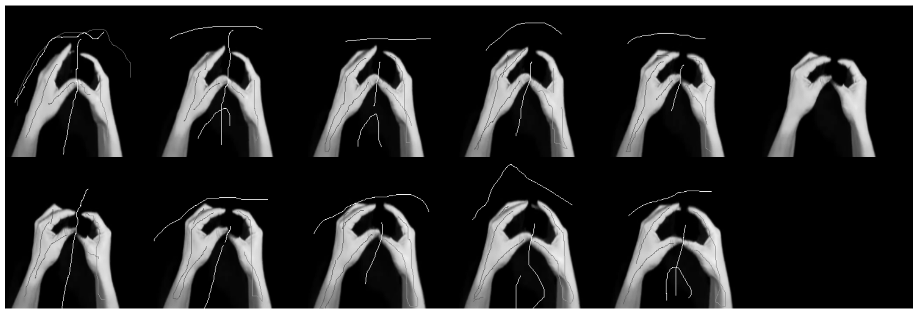


Figure 2.6: Video frames contain two hands

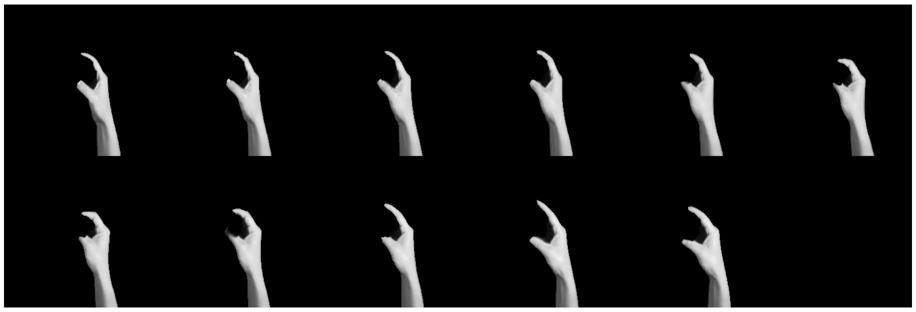


Figure 2.7: Right-hand extraction

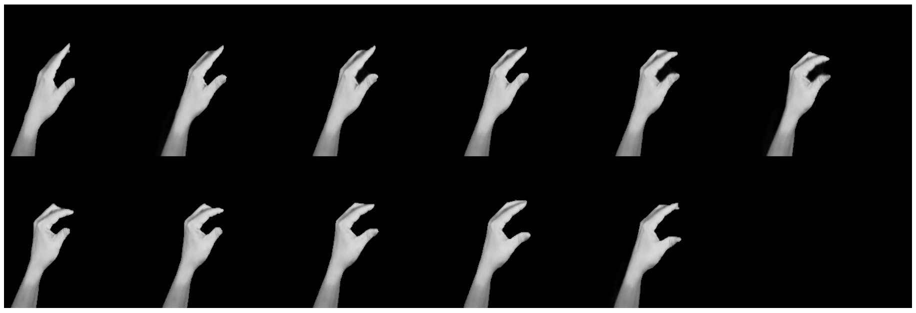


Figure 2.8: Left-hand extraction

Chapter 3

Motion Transformation and Descriptors

In this chapter we will show some known visual motion transformations. The visual motion Descriptor is one of the important features of objects in the video analysis, it gives information about the moving objects in a video, we are interested in this objects.

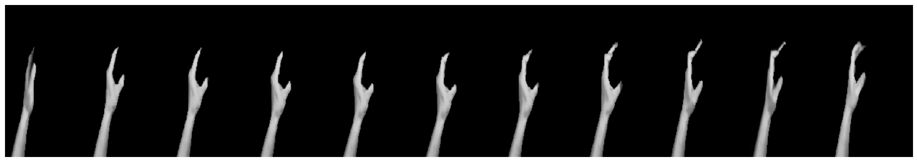


Figure 3.1: Example of a Pressure for Hard Object gesture video

3.1 Optical flow

Optical Flow (OF)[**21**] is the 2D motion field, $u(x, y); v(x, y)$, for each point (x, y) in an image. Optic flow algorithms measure the displacement of pixels between frames, this displacement gives important information as feature descriptor; also it considers as a good technique for $3D$ image segmentation 'video', because it retrieves the moving objects in a video. The basic assumption made is that pixel intensities usually are very smoothly between frames. Effectively the same pixel can be found between two frames by its intensity value. Let $I(x, y, t)$ be the image pixel intensity at time t; at some elapsed time δ_t we can write the following image constraint equation:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + \cdots$$

Where dots represents higher order terms that usually are excluded. Supposing an object is at position (x, y) at time t and at time $t + \Delta t$ has moved in space by $(\Delta x, \Delta y)$. Assume that intensity level does not change over time we can write the equation as :

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t)$$

And therefore,

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + \cdots$$

which is called the optical flow equation (***OFE***) and conventionally written as:

$$I_x u + I_y v + I_t = 0$$

where I_x, I_y and I_t are spatial and temporal derivatives of intensities which can be computed for every frame. Also $u = \frac{d_x}{d_t}$, $v = \frac{d_y}{d_t}$. This calls the optical flow equation and forms the basis of a large number of the first order algorithms(where the small motion assumption is valid). We will show here one algorithm, namely the "*Horn and Schunk algorithm*" based on the optical flow.

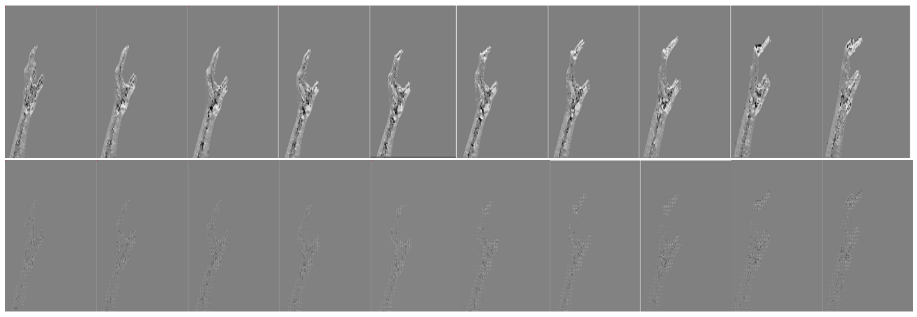


Figure 3.2: Result of applying the Horn and Schunk algorithm on the Pressure for Hard Object example above, with different block size(1,3),lambda=0.1, neighbors=8.

3.1.1 Horn and Schunk algorithm

The equation above gives only one equation for two variables (or if there are N pixels in the image, it gives N equations for $2N$ variables) and hence requires additional constraints to make it well-posed. Horn and Schunk **[21]** proposed to use the *smoothness constraint, i.e.* nearby points on an object move with similar x and y velocities, or that $\|\nabla u\|^2, \|\nabla v\|^2$ is small. Thus they proposed to find u, v to minimize the following energy functional

$$E(u, v) = \int_\Omega [(I_x u + T_y v + T_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2)]dxdy$$

Where λ controls the weight given to the smoothness constraint and Ω denotes the image domain. We assume that u and v are known (zero) at the image boundaries. E is functional now, since E is a function of u, v and their derivatives. We use calculus of variations to perform the minimization, defining the Lagrangian as:

$$L(u(x,y), v(x,y)) = [(I_x u + T_y v + T_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2)]dxdy$$

and using Calculus of Variations, we get:

$$\nabla_u E = \frac{\partial L}{\partial u} - \frac{\partial}{\partial x}\frac{\partial L}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial u_y}$$

$$\nabla_u E = I_x^2 u + I_x I_y v - I_x I_t - \lambda \nabla^2 u$$
$$\nabla_u E = I_y^2 v + I_x I_y u - I_y I_t - \lambda \nabla^2 v$$

where $\nabla^2 u = u_{xx} + u_{yy}$. A necessary condition for E to be minimized is given by $\nabla_u E = \nabla_v E = 0$. This is called the Euler-Lagrange equation[**reference**] and can be re-arranged to give

$$I_x^2 u + I_x I_y v = \lambda \nabla^2 u - I_x I_t$$

$$I_y^2 v + I_x I_y u = \lambda \nabla^2 v - I_y I_t$$

To solve the above equations for discrete pixels, the Laplacian can be approximated by their discrete central approximation, $\nabla^2 u(x,y) \approx 4(\overline{u}(x,y) - u(x,y))$ where $\overline{u}(x,y) = \frac{1}{4}(u(x-1,y) + u(x+1,y) + u(x,y-1) + u(x,y+1))$ This gives $2N$ equations in $2N$ variables: defining $\alpha = 4\lambda$, we get

$$(I_x^2 + \alpha)u + I_x I_y v = \alpha\overline{u} - I_x I_t$$

$$(I_y^2 + \alpha)u + I_x I_y u = \alpha\overline{v} - I_y I_t$$

This can in principle be solved directly, but will require inverting a very large, $2N \times 2N$, matrix. A faster method is to obtain an iterative solution using the Gauss-Seidel method which takes advantage of the sparseness of the matrix. Thus to summarize the algorithm:

Algorithm

1. *At iteration $n = 0$, start with an initial guess of u, v*
2. *Update using:*

$$u^{n+1} = \overline{u}^n - I_x \tfrac{I_x \overline{u}^2 + I_y \overline{v}^2 + I_t}{\alpha + I_x^2 + I_y^2}$$
$$v^{n+1} = \overline{v}^n - I_x \tfrac{I_x \overline{u}^2 + I_y \overline{v}^2 + I_t}{\alpha + I_x^2 + I_y^2}$$

3. *Stop when E does not decrease much or equivalently $I_x \overline{u}^n + I_y \overline{v}^n + I_t$ is small.*

3.2 Motion History Image

Motion Energy Images (MEI) and Motion History Images (MHI) are introduced to capture motion information in images. They encode, respectively, where motion occurred and the history of motion occurrences in the image. Pixel values are therefore binary values (MEI) encoding motion occurrence at a pixel; (MHI) encoding how recently motion occurred at a pixel. More formally, consider the binary-valued function $D(x; y; t)$, $D = 1$ indicating motion at time t and location $(x; y)$, then the MHI function is defined by:

$$H_\tau(x, y, t) = \begin{cases} \tau & D(x, y, t) = 1 \\ \max((H_\tau(x, y, t-1), 0) & \textit{Otherwise} \end{cases}$$

Where τ is the maximum duration where a motion is stored.

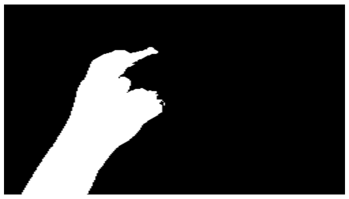


Figure 3.3: Result of applying the MHI filter on the Pressure for Hard Object example above.

3.3 Motion History Image Density per Time

MHIDT captures the motion information in images. It encodes where history of motion occurrences, in the image. This is Done by, calculation of the differences between each two sequent frames.
Let $D(x, y, t)$ be a binary video, we calculate MHIDT by:

$$H_\tau = D(x, y, t) - D(x, y, t-1)$$

Where $(t = 1 \cdots n)$ is the maximum duration where the motion is stored.

3.4 Motion History Image Density

MHID uses the same technique as MHI with one change, which is the counting of how many times the pixel belongs to the object in a video. It takes a video as input and returns 2D image which represents the historical information about the video.
Here, first we binaraize the video and then apply the MHID filter, this filter fills each pixel by a value equal to how many times this pixel is white in the video *(it belongs to the object).*

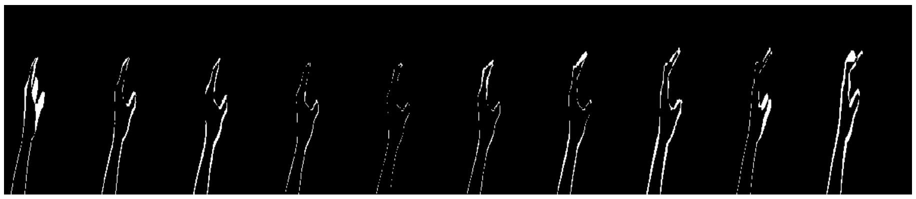


Figure 3.4: Result of applying the MHIDT filter on the Pressure for Hard Object example above.

$$H(x,y) = \sum_{z=0}^{Z} Video(z,x,y)$$
$$\textit{Where video is binary}$$

Then we delete all the pixels that have the value which is equal to Z (video.Depth()); this means that these pixels were never changed, they are white in all the frames in a video. As Final result of this filter we get an image with maximum $Z-1$ labels, as it seen in figure 3.5.

```
if ($H(x,y) = Video.Depth()$) then
    $H(x,y) = 0$
end if
```

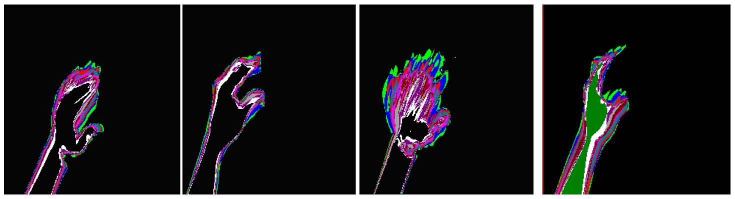


Figure 3.5: Result of applying the MHID filter.

This filter will be used in calculating the 2D moments. We calculate the 2D moments on the MHID images of the videos.

Chapter 4

Visual Feature Extraction (Shape descriptions)

In general terms, shape descriptors are a set of numbers that are found to describe a shape in compact form. A shape descriptor should ideally be a simplification of it's representative region but still hold enough information so that different shapes are discriminated. Usually it either describes the shape boundary or the image region. In our approach we use the features based region description. Here we discuss the feature vector extracted from our videos; there are many ways and many feature extraction methods that can represent our objects "hands" in a video, such as moments or historical histogram. We tested the Geometrical, Zernike, Legendre moments and the histogram.

4.1 Historical histogram

In some cases, possibly the most useful features of describing the digital images is the histogram. One disadvantage of using the histogram as feature descriptor in our work is that we are interested in the movement of the hand in the video, a normal histogram does not take into account the movement of the objects. To make the histogram more useful in describing the movement in the video we apply it on the MHID image, which represent the historical motion of the hand in the video. Then, we get N features which is equal to the number of frames in the video, ($N = Video.Depth() = 11$). The histogram then will range in 11 color levels$N = 11$, each level's value represents the number of pixels which have been belonging to the object in the video.

4.2 Gabore transformation

The Gabor Transformation has been found to be very useful for image analysis and compression tasks. The Gabor elementary functions are a set of overlapping functions and not mutually orthogonal, that makes its Implementation to be very complicated and time consuming. An important property of Gabor elementary functions is their achievement of the theoretical lower bound of joint un-certainty in the two conjoint domains, space and frequency. They achieve the maximum possible joint resolution in the conjoint visual space and spatial frequency domain. Some properties of the visual system, such as spatial localization, orientation selectivity and spatial frequency selectivity can also be modeled in terms of these functions. Although, it has been shown that the Gabor decomposition reduces the low-order entropy of the data. This means that the Gabor transform has the beneficial property of decorrelation. Therefore when the Gabor transform is used for image compression, in which the properties of the visual system can be incorporated into an image coding scheme, a high data compression ratio may be achieved.

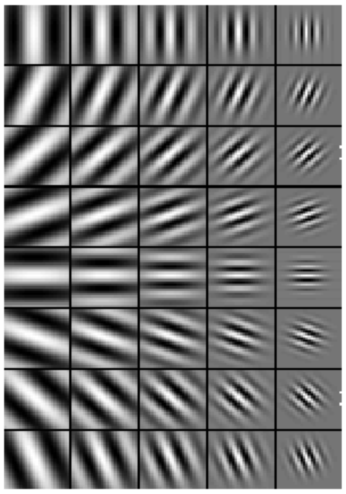


Figure 4.1: The Gabor wavelets.

Gabor functions act as low-level oriented edge and texture descriptors and are sensitive to different frequencies and scale information. The Gabor process takes as input and image and two parameters, orientation and scale. The 2D Gabor function (g) is the product of the 2D Gaussian and the complex exponential function. the general expression is given by:

$$g_{\theta,\lambda,\sigma_1,\sigma_2}(x,y) = Exp\Big\{ \ -\tfrac{1}{2(x,y)M(xy)^T} \ \Big\} Exp\big\{ \ \tfrac{j\pi}{\lambda}(x\cos\theta + y\sin\theta) \ \big\}$$

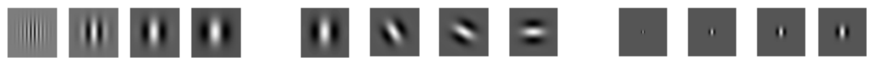
where $M = diag(\sigma_1^{-2}, \sigma_2^{-2})$, θ represents the orientation, λ is the wavelength and σ_1 and σ_2 represent the scale at orthogonal directions. When the Gaussian part is symmetric, we obtain the isotropic Gabor function:

$$g_{\theta,\lambda,\sigma_1,\sigma_2}(x,y) = Exp\{-\frac{x^2+y^2}{2\sigma^2}\}Exp\{\frac{j\pi}{\lambda}(x\cos\theta + y\sin\theta)\}$$

However, with this parametrization the Gabor function does not scale uniformly, when σ changes. It is preferable to use $\gamma = \lambda/\sigma$ instead of λ so that a change in σ corresponds to a true scale change in the Gabor function. The Equation will then be:

$$g_{\theta,\lambda,\sigma_1,\sigma_2}(x,y) = Exp\left\{ -\tfrac{x^2+y^2}{2\sigma^2} \right\} Exp\left\{ \tfrac{j\pi}{\gamma\sigma}(x\cos\theta + y\sin\theta) \right\}$$

By selectively changing each parameter of the Gabor function, we can 'tune' the filter to particular patterns arising in the images. Figure.4.2 shows the illustration of the variation parameters of(γ,θ,σ) in the shape of the Gabor function. By convolving the Gabor function with image $I(x,y)$ we can evaluate



(a) $\gamma = \{1/2, 3/2, 5/2, 7/2\}$ (b) $\theta = \{0, \pi/6, \pi/3, \pi/2\}$ (c) $\sigma = \{4, 8, 12, 16\}$

Figure 4.2: Examples of Gabor functions. Each sub-figure shows the real part of Gabor function for different values of γ, θ and σ

their similarity. We define the Gabor response at point (x,y) as:

$$G_{\theta,\lambda,\sigma}(x,y) = (I * g_{\theta,\lambda,\sigma})(x_0,y_0) = \int I(x,y)g_{\theta,\lambda,\sigma}(x_0-x, y_0-y)dxdy$$

where $*$ represents convolution. The Gabor response then can emphasize three types of characteristics in the image: edge orientation, texture orientation and the combination of both. We must vary the parameters σ, θ and γ in order to emphasize different types of image characteristics. The variation of θ changes the sensitivity to edge and texture orientations; the variation of σ will change the scale; the variation of γ the sensitivity to high/low frequencies.

4.3 The Log-Polar transformation

Input image is generally represented as a collection of pixel points on the Cartesian coordinates. Here we take the origin at the the middle pixel in the width and the height of the image. Then the Log-Polar image can be constructed by the following transformation of the coordinates, the point (x,y) on the Cartesian coordinates is transformed into the point $(\rho = \sqrt{(x^2+y^2)}, \theta = \arctan(\frac{y}{x}))$ on the Polar coordinates. then the point on the Polar coordinates is transformed into the point $(\log(\rho), \theta)$ on the Log-Polar coordinates by taking the logarithm

of the scale ρ.
From the mathematical point of view the log-polar mapping can be expressed as a transformation between the polar plane(ρ, θ) (retinal plane), the logpolar plane(ξ, η) (cortical plane) and the Cartesian plane(x, y) (image plane), as follows:

$$\begin{cases} \eta = q \cdot \theta \\ \xi = log_a \frac{\rho}{\rho_0} \end{cases}$$

where ρ_0 is the radius of the innermost circle, $1/q$ is the minimum angular resolution of the log-polar layout and (ρ, θ) are the polar co-ordinates. These are related to the conventional Cartesian reference system by:

$$\begin{cases} x = \rho \cos \theta \\ y = \rho \sin \theta \end{cases}$$

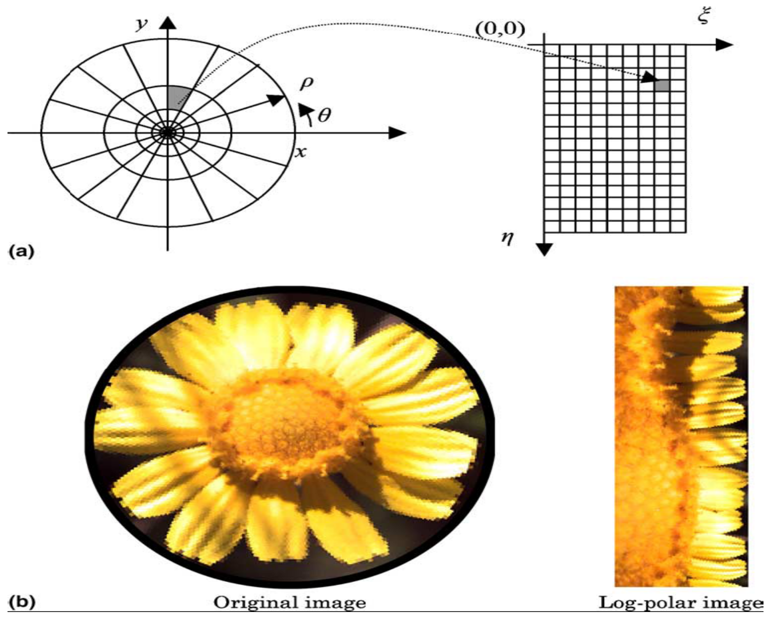


Figure 4.3: The Log-Polar transformation: (a) graphical illustration and (b) example of a real image.

Fig. 2 illustrates the log-polar layout as derived from the last two equations above. In particular, in Fig. 2a the grid on the left represents a standard Cartesian image mapped according to the transformation between logpolar plane and the Cartesian plane. The plot on the right shows the corresponding log-polar image. Fig. 2b presents a Cartesian image and its log-polar counterpart. It is worth noting that the flower petals, that have a polar structure, are mapped horizontally in the log-polar image. Circles, on the other hand, are mapped vertically. Furthermore, the stamens that lie in the center of the image of the flower, occupy about half of the corresponding log-polar image.

Note: *Although we will not apply these transformations (Gabor and Log-Polar) in our thesis, we would like to have them derived in order to have some standard reference in the future.*

4.4 Moments

Moments are a measure of the spatial distribution of 'mass', the shape of an object. Objects in a binary image are represented as a set of white pixels (in $2D$) and voxels (in $3D$). Videos are a $2D + t$ (time) image, so we can represent it as a $3D$ image with Depth equal to t.

4.4.1 Moment invariants

Moment invariants[**13**] are important shape descriptors in computer vision. There are two types of shape descriptors: contour-based shape descriptors and region-based shape descriptors. Regular moment invariants are one of the most popular and widely used contour-based shape descriptors. Here we will show the region-based shape descriptor.

2D HU Moment invariants

Hu described a set of seven moments[**13**] that are **rotation, scaling and translation** invariant Two-dimensional moments of a digitally sampled image that has gray function $f(x, y)$,are given by:

$$M_{pq} = \int \int f(x, y) x^p y^q \ dxdy$$

p, q represent the order of the moments and $f(x, y)$ denotes to the pixel's value at position(x, y). In digitalization we change the integration to summation. The moments $f(x, y)$ translated by an amount (a, b), are defined as,

$$\mu_{pq} = \int \int f(x, y)(a - \overline{X})^p (b - \overline{Y})^q \ dxdy$$

where $\overline{X} = \frac{M_{10}}{M_{00}}$, $\overline{Y} = \frac{M_{01}}{M_{00}}$, M_{00} denotes the area of the object, and $(\overline{X}, \overline{Y})$ denotes to the center of the object. Translation, rotation and scaling are represented by φ_i, where $i = 1..7$.
The $2D$ Hu feature descriptors are calculated from the videos by applying the $2D$ Hu invariant moments on the historical images. Seven moments are extracted as a feature vector:
$\varphi_1 = \mu_{20} + \mu_{02}$
$\varphi_2 = (\mu_{20} + \mu_{02})^2 + 4\mu_{11}^2$
$\varphi_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$

$\varphi_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} - \mu_{03})^2$
$\varphi_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} - \mu_{03})^2] + (3\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} - \mu_{03})^2]$
$\varphi_6 = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} - \mu_{03})^2] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$
$\varphi_7 = (3\mu_{12} - \mu_{30})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} - \mu_{03})^2] + (3\mu_{12} - \mu_{03})(\mu_{21} - \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} - \mu_{03})^2]$

3D Geometrical Moment

It is possible to compute moment invariants of 3D point distributions which are invariant to translation and rotation, in the same manner as 2D moment invariance. A set of 14 moments derived by HU gives information about region-based shape descriptor, which are rotation, scaling and translation invariant in a 3D dimensions.
Let (x,y,t) be a binary video, that means the values of its voxels are equal to one for the voxels belonging to the object and equal to zero for the background. We can define the moment as:

$$A_{pqr} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q t^r \, dxdydt$$

A_{000} represents the area of the object and $(A_{100}, A_{010}, A_{001})$ are the center coordinates of the object.

$$M_{pqr} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \left[\frac{x - A_{100}}{A_{200}^{1/4} * A_{020}^{1/4}}\right]^p \left[\frac{y - A_{010}}{A_{200}^{1/4} * A_{020}^{1/4}}\right]^q \left[\frac{t - A_{001}}{A_{200}^{1/2}}\right]^r dxdydt$$

In discrete, the integration is changed to summation. The 3D geometrical feature descriptors are calculated from our videos by applying the 3D geometrical moments directly on the videos. 14 Moments[**15**] are extracted as a feature vector:

$$M_{3d} = \begin{cases} M_{200}, M_{011}, M_{101}, M_{110}, M_{300}, M_{030}, M_{003}, \\ M_{210}, M_{201}, M_{120}, M_{021}, M_{102}, M_{012}, M_{111} \end{cases}$$

4.4.2 2D Zernike Moments

Zernike moments[**13**] are a class of orthogonal moments and have been shown effective in terms of image representation. They are based on the Zernike orthogonal Zernike radial polynomial. These moments are effectively used in pattern recognition since their rotational invariants can be easy constructed to an arbitrary order. Although higher order moments carry more fine details of an image, they are also susceptible for noise.
The Zernike polynomials are an orthogonal set of complex valued polynomials and they are defined as:

$$V_{nm}(x, y) = R_{nm}(x, y)e^{j*m*\arctan(y/x)}$$

where $x^2+y^2 \leq 1, \cdots, n \geq 0, \mid m \mid\mid \leq n$ and $n- \mid m \mid$ is even. Radial polynomials $\{R\}$ are defined as:

$$R_{mn}(x,y) = \sum_{s=0}^{(n-|m|)/2} S_{n,|m|,s}(x^2+y^2)^{(n-2s)/2}$$

$$S_{n,|m|,s} = (-1)^s \frac{(n-s)!}{s!(\frac{n+|m|}{2}-s)!(\frac{n-|m|}{2}-s)}$$

The complex Zernike moments of order n and repetition m are given by:

$$ZMI_{mn} = \frac{n+1}{\pi} \sum_X \sum_Y f(x,y) V^*_{nm}(x,y)$$

Zernike feature descriptors are calculated on our videos by applying the Zernike moments on the historical images of the videos. Zernike moments of order eight are extracted as a feature descriptor.[**16**]

$$Zernike2D = \begin{cases} Z_{11} \\ Z_{20}+Z_{22} \\ Z_{31}+Z_{33} \\ Z_{40}+Z_{42}+Z_{44} \\ Z_{51}+Z_{53}+Z_{55} \\ Z_{60}+Z_{62}+Z_{64}+Z_{66} \\ Z_{71}+Z_{73}+Z_{75}+Z_{77} \\ Z_{80}+Z_{82}+Z_{84}+Z_{86}+Z_{88} \end{cases}$$

4.4.3 2D Legendre Moments

Legendre moments[**13**] are using the Legendre polynomials as kern function, and they belong to a class of orthogonal moments. They can be used to attain the near zero value of redundancy measure in a set of moments function so that the moments are corresponding to independent characteristics of the image. The two-dimensional Legendre moments of order(p,q), with image intensity function $f(x,y)$, are defined as:

$$L_{pq} = \frac{(2p+1)(2q+1)}{4} \int_X \int_Y P_p(x) P_q(y) f(x,y) dxdy$$

where the Legendre polynomial $P_p(x)$ of order p is given by:

$$P_p(x) = \sum_{k=1}^{p} \left\{ (-1)^{\frac{p-k}{2}} \frac{1}{2^p} \frac{(p+k)!x^k}{\left[\frac{p-k}{2}\right]! \left[\frac{p+k}{2}\right]! k!} \right\}_{(p-k)=even}$$

The recurrence relation of Legendre polynomials $P_p(x)$ is given as:

$$P_p(x) = \frac{(2p-1)P_{p-1}(x) - (p-1)P_{p-2}(x)}{p}$$

where $P_0(x) = 1, P_1(x) = x$, and $p > 1$. Since the region of definition of Legendre polynomials is the interior of$\{1, -1\}$, a square of NxN pixels with intensity function $f(I, j), 0 <= I, j <= (N - 1)$, is scaled in the region of $(-1 < x, y < 1)$, we can now define a new equation for L_{pq}:

$$L_{pq} = \lambda_{pq} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P_p(x_i) P_q(y_j) f(i, j)$$

where the normalization constant is:

$$\lambda_{pq} = \frac{(2p + 1)(2q + 1)}{N^2}$$

and x_i and y_j denote the normalized pixel coordinates in the range of $\{-1, 1\}$, which are given by:

$$x_i = \frac{2!}{N - 1} - 1, \; y_j = \frac{2!}{N - 1} - 1$$

Legendre feature descriptors are calculated on the historical images in the same way as Zernike. Nine Legendre moments are extracted as a feature descriptor.

$$Legendre2D = \begin{cases} L_{11} \\ L_{20} + L_{22} \\ L_{31} + L_{33} \\ L_{40} + L_{42} + L_{44} \\ L_{51} + L_{53} + L_{55} \\ L_{60} + L_{62} + L_{64} + L_{66} \\ L_{71} + L_{73} + L_{75} + L_{77} \\ L_{80} + L_{82} + L_{84} + L_{86} + L_{88} \end{cases}$$

Chapter 5

Categorization and Recognition by Diffusion Maps

5.1 Categorization

Diffusion maps (DM)**[17]**[**18**] are based on defining the Markov random walk**[17]** on the graph of data. By performing the random walk for a number of time steps, a measure for proximity of the data points is obtained. Using this measure, the so-called diffusion distance is defined. In diffusion maps the graph of the data is constructed first. Let $G = (V, E)$ be an undirected graph with vertex set $V = v_1, \cdots, v_n$. In the following we assume that the graph G is weighted, we compute the weights of the edges in the graph using the Gaussian kernel function, leading to the similarity matrix W of the graph G.

$$W_{ij} = e^{-\frac{d_{ij}}{2\sigma^2}}$$

Where σ indicates the variance of the Gaussian and d_{ij} denotes to the distance between v_i and v_j.The degree of a vertex $v_i \in V$ is defined as :

$$D_i = \sum_{i=1}^{n} w_{ij}$$

We define the diagonal matrix D by: $D_{ii} = D(v_i, v_i) = d_i$, and $D_{ij} = 0$ for $i \neq j$. Defining the matrix L such as:

$$L_{ij} = L(v_i, v_j) = \begin{cases} d_i - W_{ii} & if\ v_i = v_j \\ -W_{ij} & if\ v_i\ and\ v_j\ are\ adjacents \\ 0 & otherwise \end{cases}$$

While the Laplacian of graph G can be defined by:

$$\Im = D^{-1/2} L D^{-1/2} \ \textit{Where } D_{ii}^{-1} \equiv 0 \textit{ if } d_i = 0$$

The transition probability of vertex v_i to vertex v_j in each step is: $P_{ij} = w_{ij}/d_i$. This defines the transition matrix P of the Markov chain.

$$\forall v_i, v_j, 0 \leq p_{ij} \leq 1 \text{ and } \sum_{j \in V} p_{ij} = 1 \text{ we also can write } P = D^{-1}W$$

Now, we define our approach of categorization based diffusion maps in graph.

```
Input: graph vertices $X = \{x_1, x_2, \cdots, x_n\} \subset \Re^d$ , $t, m, \varepsilon$
Output: Clusters $A_1, \cdots, A_k$ with $A_i = \{j | y_j \in C_i$
Construct a matrix of similarity. Let $W$ be its weighted adjacency matrix.
    $W_{ij} = e^{\frac{\|x_i - x_j\|^2}{\varepsilon}}$
Normalization by using the Laplace-Beltrami method:
    $\widetilde{W_{ij}} = W_{ij}/(\sqrt{d_i d_j})$
Calculating the transition matrix: $p_{ij} = \widetilde{W_{ij}}/(\sqrt{\widetilde{d_i}\widetilde{d_j}})$ With $d_i = \sum_{i=0}^{n-1} w_{ij}$
Diagonalization of matrix $P$
Diffusion space:
Compute the first $k$ eigenvectors $v_1, \cdots, v_k$ of $P$
Normalize the eigenvectors, dividing each row by its first element value
$Y$= sorting the vectors by $\lambda_1, \lambda_2, \lambda_3$
Cluster the points $(y_i)_{i=1,\cdots,n}$ in $R_k$ with the k-means algorithm into clusters $C_1, \cdots, C_k$
```

The first normalization of the similarity matrix allows finding one independent representation of the distribution. The spectral decomposition of matrix P gives a set of Eigen values $1 = |\lambda_0| \geq |\lambda_1| \geq \cdots \geq |\lambda_n| \geq 0$, which generates a set of Eigen vectors $\{\varphi_0, \varphi_1, \cdots, \varphi_n\}$, solution of $P\varphi_m = \lambda_m^t \varphi_m$ such as; we can define a family of diffusion distances $\{D_t\}_{t \geq 1}$ as:

$$D_t^2(x, y) = \sum_{j \geq 0} \lambda_j^t (\varphi_j(x) - \varphi_j(y))^2$$

Where t is a scale parameter controls the sensitivity of diffusion distance D_t to φ_j. Consider the following transformation of $\{\psi_t\}_{t \geq 1}$:
$\psi_t : \Re^n \to \Re^{m(t)}$
$x \to \psi_t(x) = (\lambda_0^{t/2}\varphi_0(x), \lambda_1^{t/2}\varphi_1(x), \lambda_2^{t/2}\varphi_2(x), \cdots, \lambda_{m(t)}^{t/2}\varphi_{m(t)}(x))^T$ with $\varphi_0(x) = (1, 1, \cdots, 1)$. transformation commonly utilizes for analysis and data reduction

in a large dimension *[Stephane Lafon]*. That allows to go from a large dimension space n to a homogeneous space dimension $m(t)$ also reduce the information such as which represent the structure properties of the graph. $m(t)$ indicates the number of which Eigen values are insignificants. Generally $m(t) <= 3$. The diffusion distance then formed as:

$$D_t^2(x,y) = \|\psi_t(x) - \psi_t(y)\|.$$

5.2 Clustering using k-mean algorithm

Clustering can be considered the most important *unsupervised learning* problem; so, as every other problem of this kind, it deals with finding a structure in a collection of unlabeled data. A loose definition of clustering could be "the process of organizing objects into groups whose members are similar in some way".
A cluster is therefore a collection of objects which are "*similar*" between them and are "*dissimilar*" to the objects belonging to other clusters. **The k-means algorithm** is an algorithm to cluster n objects based on attributes into k partitions, $k < n$. The objective it tries to achieve is to minimize total intra-cluster variance, or, the squared error function:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

where there are k clusters S_i, $i = 1, 2, \cdots, k$, and μ_i is the centroid or mean point of all the points $x_j \in S_i$.
The K-means algorithm will do the three steps below until convergence
Iterate until stable (no object moves group):

1. Determine the centroid coordinate.
2. Determine the distance of each object to the centroids
3. Group the object based on minimum distance

5.3 Recognition by Diffusion maps

5.3.1 Kernel Methods basics

Let $\Omega = \{x_1, x_2, \cdots, x_n\} \subset \Re^d$ be the set of training points. The kernel is a function $k : \Omega \times \Omega \to \Re$ such that there exist a mapping $\varphi : \Omega \to H$, where H is a Hilbert space and the following inner-product relationship holds $k(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle, \;\; i, j = 1, \cdots, n$

Let K be the matrix containing the kernel values,$K_{ij} = k(x_i, x_j)$. If this matrix is semi definite positive, then k is a kernel over the set Ω. A mapping satisfying the dot product property can be found by the Eigen-decomposition of the kernel matrix K: $K = U \wedge U^T = U \wedge^{1/2} (U \wedge^{1/2})^T$ Where U is the matrix whose columns are the eigenvectors $\phi_i, \ i = 1, \cdots, n, \ and \wedge = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$ is the diagonal matrix of the Eigen values in decreasing order. If we define $\varphi(x_i)$ to be the i^{th} row of $U \wedge^{1/2}$, and since the Eigen vectors are non-negative (*positive semi definite matrix*), we obtain the desired mapping:

$$\varphi(x_i) = [\sqrt{\lambda_1}\phi_1(x_i), \sqrt{\lambda_2}\phi_2(x_i), \cdots, \sqrt{\lambda_n}\phi_n(x_i)]$$

The kernel function can then be considered as a generalization of the dot product, and therefore it is a measure of similarity between the input points.

5.3.2 The Nystrom extension

Let $x \in \Re^d$ be a new input point not in the training set. The Nystrom extension[**19**], states that the j^{th} coordinate of the kernel mapping ϕ for this point can be approximated as:

$$\varphi_j(x) = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} k(x, x_i)\phi_j(x_i) \quad j = 1, 2, \cdots, n$$

or in vector form:

$$\varphi(x) = \tfrac{1}{\sqrt{\wedge}} U^T k_x$$

where $k_x = [k(x, x_1), k(x, x_2), \cdots, k(x, x_n)]$, and $\frac{1}{\sqrt{\wedge}}$ stands for $(\sqrt{\wedge})^{-1} = diag(\frac{1}{\sqrt{\lambda_1}}, \cdots, \frac{1}{\sqrt{\lambda_n}})$. In other words, the new point x is mapped as a weighted linear combination of the corresponding maps for the training points x_i. The weights are given, modulo normalization by the Eigen values, by the kernel relationship $k(x, x_i)$ representing the similarity between x and x_i. Observe that while extending the mapping, we also need to extend the kernel. This is straightforward when the kernel defined over Ω is simply a known function defined in the ambient space $\Re^d$. In other cases the extension of the kernel is not trivial.

Chapter 6

Pattern Recognition

Pattern recognition is the study of how machines can observe the environment, learn how to distinguish patterns of interest and make decisions about categories of the pattern. In our approach the patterns are videos of one moving hand, the movement of the hand gives information about what is the action. So we would like to give the system the ability to recognize the action done by the hand. Several approaches are supposed here, the recognition part of our approach based neural networks.

6.1 Neural networks

An Artificial Neural Networks (ANNs)[**22**] process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural networks learn by example as like as humans. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Neural networks, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an *"expert"* in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Other advantages include:

1. ***Adaptive learning***: An ability to learn how to do tasks based on the data given for training or initial experience.

2. ***Self-Organization***: An ANN can create its own organization or repre-

sentation of the information it receives during learning time.

3. ***Real Time Operation***: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

4. ***Fault Tolerance via Redundant Information Coding***: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

6.1.1 Simple neuron

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

6.1.2 A more complicated neuron

The previous neuron doesn't do anything that conventional computers don't do already. A more sophisticated neuron is the McCulloch and Pitts model (MCP). The difference from the previous model is that the inputs are 'weighted'; the effect that each input has at decision making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a pre-set threshold value, the neuron fires. In any other case the neuron does not fire. In mathematical terms, the neuron fires if and only if;

$$X_1W_1 + X_2W_2 + \cdots > T$$

The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to *'adapt'*; the most used ones are the Delta rule and the back error propagation. The former is used in feed-forward networks and the latter in feedback networks.

6.1.3 Architecture of neural networks

Feed-forward networks

Feed-forward ANNs allow signals to travel only in one way; from input to output. There is no feedback *(loops)* i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

Feedback networks

Feedback networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations.

6.1.4 Network layers

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of *"input"* units is connected to a layer of *"hidden"* units, which is connected to a layer of *"output"* units.

- The activity of the input units represents the raw information that is fed into the network.
- The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.
- The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering.

6.1.5 The Learning Process

The memorization of patterns and the subsequent response of the network can be categorized into two general paradigms:

1. **Associative Mapping** in which the network learns to produce a particular pattern on the set of input units whenever another particular pattern is applied on the set of input units. The associative mapping can generally be broken down into two mechanisms:

- *Auto-association*: an input pattern is associated with itself and the states of input and output units coincide. This is used to provide pattern competition, ie to produce a pattern whenever a portion of it or a distorted pattern is presented. In the second case, the network actually stores pairs of patterns building an association between two sets of patterns.
- *Hetero-association*: is related to two recall mechanisms:
  (a) *Nearest-neighbor recall*, where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and
  (b) *Interpolative recall*, where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented. Yet another paradigm, which is a variant associative mapping is classification, ie when there is a fixed set of categories into which the input patterns are to be classified.

2. **Regularity Detection** in which units learn to respond to particular properties of the input patterns. Whereas in associative mapping the network stores the relationships among patterns, in regularity detection the response of each unit has a particular '*meaning*'. This type of learning mechanism is essential for feature discovery and knowledge representation.

Every neural network processes knowledge which is contained in the values of the connections weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. Information is stored in the weight matrix W of a neural network. Learning is the determination of the weights. Following the way learning is performed, we can distinguish two major categories of neural networks:

1. fixed networks in which the weights cannot be changed, ie $dW/dt = 0$. In such networks, the weights are fixed a priori according to the problem to solve.

2. adaptive networks which are able to change their weights, ie $dW/dt \neq 0$.

All learning methods used for adaptive neural networks can be classified into two major categories:

1. *Supervised learning* which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. An important issue concerning supervised learning is the problem of error convergence, ie the minimization of error between the desired and computed unit values. The aim is to determine a set of weights which minimizes the error. One well-known

method, which is common to many learning paradigms, is the least mean square (LMS) convergence.

2. *Unsupervised learning* uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Paradigms of unsupervised learning are Hebbian learning and competitive learning. We say that a neural network learns off-line if the learning phase and the operation phase are distinct. A neural network learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line.

6.1.6 Transfer Function

The behavior of an ANN (Artificial Neural Network) depends on both the weights and the input-output function *(transfer function)* that is specified for the units. This function typically falls into one of three categories:

1. *Linear units*, the output activity is proportional to the total weighted output.
2. *Threshold units*, the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.
3. *Sigmoid units*, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.

To make a neural network that performs some specific task, we must choose how the units are connected to one another, and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence.

6.1.7 The Back-Propagation Algorithm

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the EW. The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection. Note that for non-linear units, the back-propagation algorithm includes an extra step. Before back-propagating, the EA must be converted into the EI, the rate at which the error changes as the total input received by a unit is changed.

Algorithm

Units are connected one to another. Connections correspond to the edges of the underlying directed graph. There is a real number associated with each connection, which is called the weight of the connection. We denote by W_{ij} the weight of the connection from unit u_i to unit u_j. It is then convenient to represent the pattern of connectivity in the network by a weight matrix W whose elements are the weights W_{ij}. Two types of connection are usually distinguished: excitatory and inhibitory. A positive weight represents an excitatory connection whereas a negative weight represents an inhibitory connection. The pattern of connectivity characterizes the architecture of the network. A unit in the output layer determines its activity by following a two step procedure.

1. First, it computes the total weighted input x_j, using the formula:

   $$X_j = \sum_j y_i Wij$$

   where y_i is the activity level of the j^{th} unit in the previous layer and W_{ij} is the weight of the connection between the i^{th} and the j^{th} unit.

2. Next, the unit calculates the activity y_j using some function of the total weighted input. Typically we use the sigmoid function:

   $$y_j = \frac{1}{1 + e^{-x_i}}$$

Once the activities of all output units have been determined, the network computes the error E, which is defined by the expression:

$$E = \frac{1}{2} \sum_i (y_i - d_i)^2$$

Where y_j is the activity level of the j^{th} unit in the top layer and d_j is the desired output of the j^{th} unit. The back-propagation algorithm steps:

1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.
5. Assign *"blame"* for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat the steps above on the neurons at the previous level, using each one's *"blame"* as its error..

Chapter 7

Experiments and Conclusion

7.1 Experiments

It is a big challenge to find out what features are the best to use in classification of the actions and the hands. For classification, we need the input vector which describes the features of the video. To find out the right features that represent represent the action and or the hand in the video, We checked some feature vectors descriptors like Histogram, Moments$2D$ *(Hu$2D$, Zernike$2D$, Legendre$2D$)* and Moments$3D$. Histogram and Moments$2D$ were calculated on the MHID image of each video, and Moments$3D$ was calculated directly on each video. As classifiers we used two approaches: the Diffusion Maps and the neural networks based back-propagation. We tested the five feature descriptors above; The results were quite good with the moments$3D$; the other feature descriptors did not give good results. Consequently we took the moments$3D$ as feature descriptor vector in our work.

We arranged our videos (after the segmentation and separation process) into two sets, the training set and the test set. The training set contains 120 videos which represent the four actions done by both the left and the right hand in the following partition: 20 videos of Pressure for Soft Object, 29 videos of Lateral Motion for Granular Object, 37 videos of Pressure for Hard Object and 34 videos of Lateral Motion for Smooth Object. The test set contains 34 new videos which represent four actions as the following partition: seven videos of PSO, seven videos of LMGO, ten videos of PHO and ten videos of LMSO, each video we have consists of 11 frames.

Using Moments$3D$ as feature descriptor with the two classifiers, we obtained the following results:

1. ***Diffusion Maps*** We tried to apply the diffusion map approach on the data-base of the videos for categorizing the 3D hand gestures. We took the

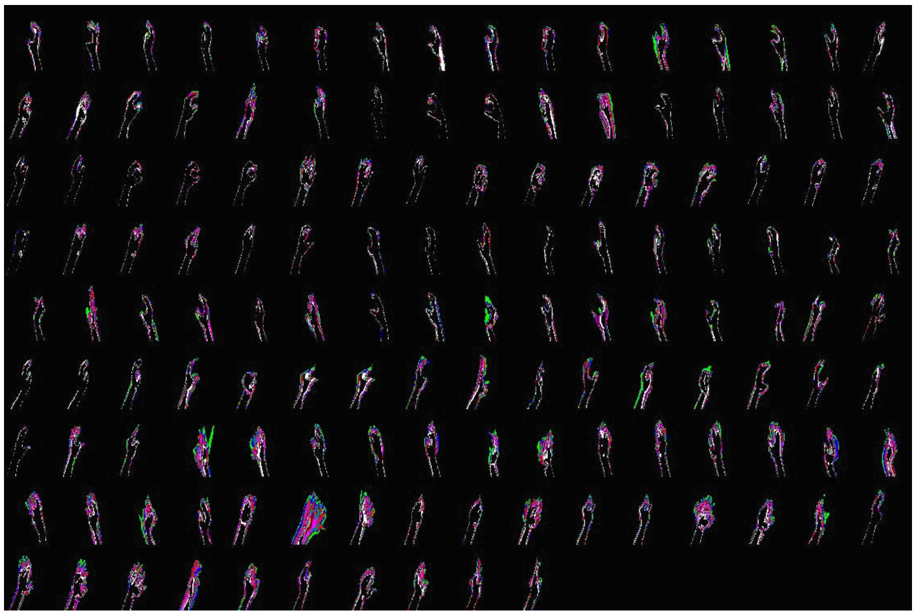


Figure 7.1: part of our database videos after MHID transformation

features based-moments as input. Here we aimed to classify the actions; The Diffusion Maps algorithm could not classify the actions but it classified the hands -with no previous knowledge- into two classes, left and right hands. This result was good for us because it classifies well which hand is presented in the video. Figure 7.2 shows results with 3D geometrical moments. As can be seen, the database is now ordered, the left hands are together as one class and the right hands are in a different class. Here the calculation was applied directly on the videos, but we show the MHID videos as result for its simplicity for the interpretation. After we apply our algorithm of diffusion maps, the result we have seen shows that this algorithm could greatly separate the hands into two classes left hand and right hand. Now we aim to recognize the hands in a new videos (videos of moving hand). To do that, we use the Nystrom extension approach as a recognizer function. We calculate the 3d-geometrical moments, the Eigen vectors and Eigen values for the training set (120 videos). For the new video we calculate its feature descriptor (3d-geometrical moments), and then the similarity with all the training set. Then we apply the Nystrom extension approach of the following function:

$$\varphi_j(x) = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} k(x, x_i)\phi_j(x_i) \quad j = 1, 2, \cdots, n$$

Here we have k as the transition vector of the similarity between the

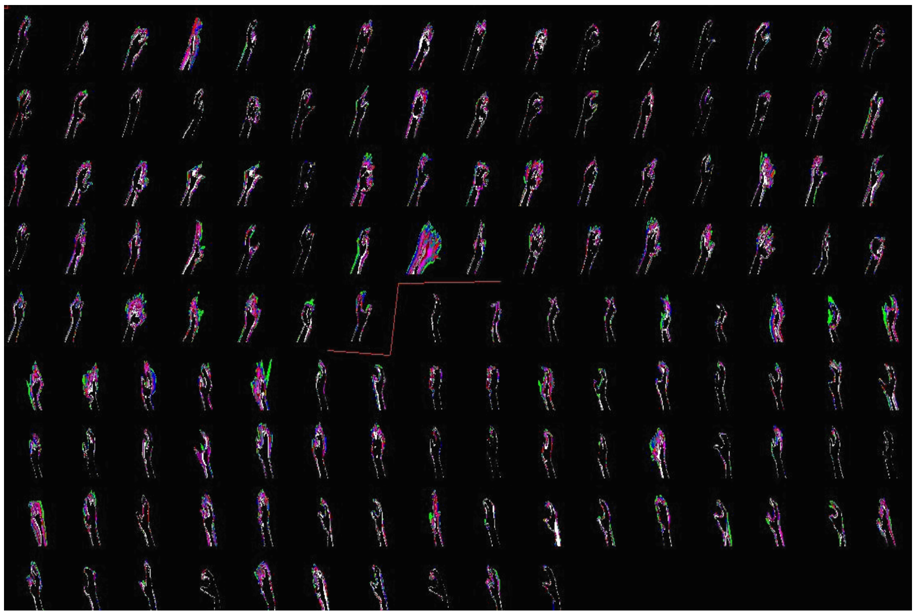


Figure 7.2: Categorization of left and right hands videos by Diffusion Maps

new video and the training set. To reduce the calculation we take only the first three Eigen values $(\lambda_1, \lambda_2, \lambda_3)$, and the first three Eigen vectors (ϕ_1, ϕ_2, ϕ_3). The result of the Nystrom extension algorithm is a vector of three elements which represent the new video. We calculate the distance between this vector and the center of the two classes, the nearest center defines which class this hand is (left or right).
The result of the recognition of the hands in these videos was 100% . Our approach gave right answers on all the videos we have

2. ***Neural Networks*** We built two neural networks based on the back-propagation algorithm. The first network answers whether the hand in a video is left or right; the second network answers what is the action or the gesture done by the hand. Both networks were trained using matlab neural networks tool, using the sigmoid activation function and Levenberg-Marquardt[**20**] learning rule.
The two networks were trained successfully with the 3D-geometrical moments as the feature descriptor of our videos composed of all hand gestures. We used the 14 geometrical moments as input for the two networks, manually we selected the output of each example in the training set.

The first neural Network ; It contains three layers, the first layer is the input layer and consist of 14 nodes (the moments$3D$); the hidden layer contains of 30 neurons. Each neuron has a weight matrix of 14

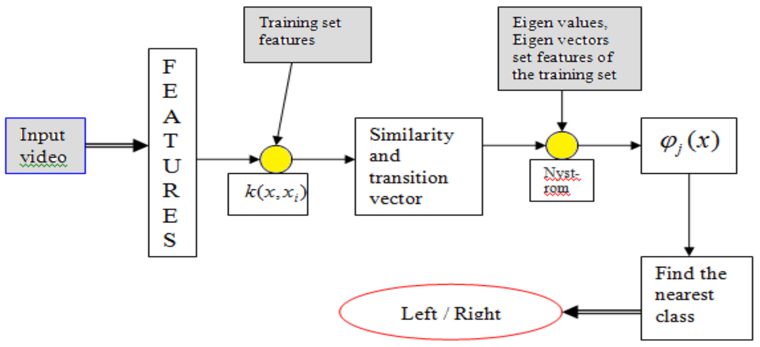


Figure 7.3: flow processing diagram for Categorization of left and right hands videos by Diffusion Maps

weights, one base and one output, and it uses the sigmoid activation function; the output layer consists of one neuron only which takes as input the outputs of the hidden layer neurons (30) and gives as output the result of which hand is moving in the video (left or right hand). For the training set, manually, we selected the out put of each video for the training process; We set (1) to the output of the right hands and (-1) of the left hands.
The network was trained successfully on the training set. It gave 100% right answers. We presented the new examples which are in the test set (34 videos, of which are 16 Left and 16 Right): it recognized the hands in these videos well, 97% right answers.

The second Neural Network ; Which is more important for us, recognizes the action done by the hands. This network also contains thee layers: the first layer is the input layer and consists of 14 nodes (the moments); the hidden layer contains of 30 neurons, each neuron has a weight matrix of 14 weights, one base and one output, and it uses the sigmoid activation function; the output layer consists of two neurons which take as input the outputs of the hidden layer neurons (30) and gives as output the result of what is the gesture or the action done by the moving hand in a video. Here we represent the output values as function in the following transformation:

1	1	***PSO***
-1	1	***LMGO***
1	-1	***PHO***
-1	-1	***LMSO***

The network was trained successfully on the training set. We presented the new examples which are in the test set (34 videos), it recognized good the gestures in these videos, resulting in 82.4% right answers.

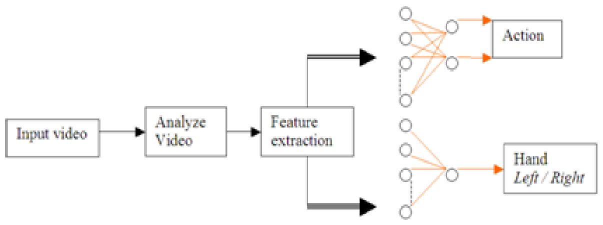
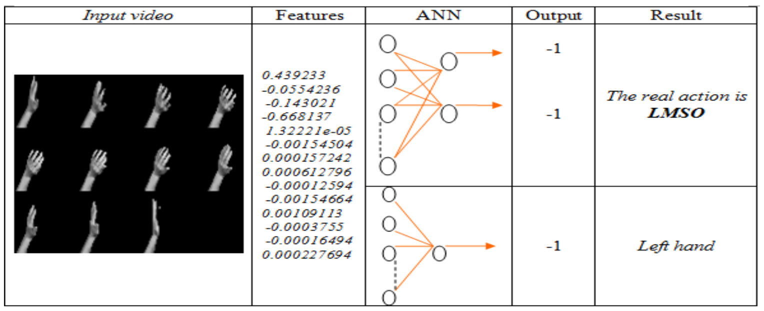


Figure 7.4: Flow processing diagram for neural networks L/R and action recognition

The result of the test set videos for the two categories (left/right and the four actions) is summarized respectively in the following table:

First Neural Network for L/R HAND	100%	*Left Hand*
	93.8%	*Right Hand*
2^{nd} Neural Network for Gesture recognition	71.4%	*PSO*
	85.7%	*LMGO*
	80%	*PHO*
	90%	*LMSO*



Input video	Features	ANN	Output	Result
	0.439233 -0.0554236 -0.143021 -0.668137 1.32221e-05 -0.00154504 0.000157242 0.000612796 -0.00012594 -0.00154664 0.00109113 -0.0003755 -0.00016494 0.000227694		-1 -1	*The real action is* ***LMSO***
			-1	*Left hand*

Figure 7.5: Example of the two neural networks

As a final result, we took the Diffusion Maps as a good approach to answer the question: Which hand is presented in the video? *(Left or Right hand)*; and the neural network to answer the question: What is the action done by this hand in the video? *(LMSO, PHO, PSO, LMGO)*.

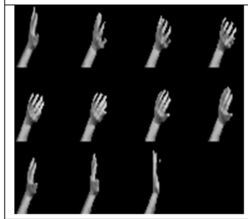
Input video	*features*	*Kernel Method*	$\varphi_j(x)$	*Result*
	*0.439233 -0.0554236 -0.143021 -0.668137 1.32221e-05 -0.00154504 0.000157242 0.000612796 -0.000125942 -0.00154664 0.00109113 -0.0003755 -0.000164941 0.000227694*	*Similarity and transition vector then calculate the Nystrom extension*	$\varphi_1(x)$ = -0.6378 $\varphi_2(x)$ = -0.1059 $\varphi_3(x)$ = -1.3213	Left hand

Figure 7.6: Example of the Diffusion Maps L/R recognition

7.2 Conclusion and Future work

A new approach based on Graphcut, 3D geometrical moments, neural networks and Diffusion Maps for hand gesture recognition is presented. The proposed framework classifies haptic properties through the video analysis of hand actions. Two objects properties have been tested: texture and consistency. For each property, two modalities were proposed. The texture of the object being explored could either be smooth or granular. Its consistency could either be hard or soft. In this algorithm, after extracting efficient dynamic hand (in 3D) by Graphcut technique and applying necessary processing on theses videos of hand gestures, robust global features' vectors are extracted, based on 3D geometrical moments. These vectors are then used to train a neural network for hand gesture recognition and to categorize the hands (left and right) by Diffusion Maps. We tested the proposed algorithms with the collection data set. The results showed a correct haptic gesture recognition rate of 82.4 percent using the neural networks based back propagation. On the other hand, the results also showed a high recognition rate of 97 percent for left/right hand recognition using the neural networks and 100 percent with the Diffusion Maps categorization. Furthermore, the proposed automatic approach is robust to traditional problems of gesture extraction and recognition. The framework can be used for interaction handicapped persons with the computer to increase their abilities.
It is a challenge to build a system which is able to recognize the gestures tracked by the camera. There are many avenues of future work for this research, including the recognition of the gestures from the live camera, improve the gesture recognition system in order to reduce the recognition errors, the recognition system should correct training data of the gesture in real time while a user performs gesture, we also plan to improve the segmentation algorithm to separate the hands in real time, and the most perspective work as future work for this research is to build a virtual reality system which gives more interactions between human and computer. This virtual reality system must be able to imitate the hand gestures taken by the camera. Also it will give more realization for the user to understand well the gestures.

Chapter 8

Extension Work, Human Motion Action Categorization and Recognition Using Diffusion Maps

8.1 Introduction

RECOGNIZING of Human Motion Actions from videos is a challenging research problem in computer vision; it is a key component in many computer vision applications, such as video surveillance, human-computer interface, video indexing and browsing, recognition of gestures, analysis of sports events, and dance choreography.

It is of relevance to both the scientific and industrial communities. Recent works in the computer vision literature have proposed a number of successful motion recognition approaches based on nonlinear manifold learning techniques. Despite significant recent developments, general human motion recognition is still an open problem.

In this research, we aim to categorize and recognize 10 human actions such as: walking, bending, jumping, jumping in the same place, running, skipping, walking a side, waving two hands, waving one hand and jacking. Then we propose a Diffusion Maps as a learning algorithm for categorizing these ten actions. Our learning algorithm is unsupervised method, we don't give any previous knowledge on our training set.

8.2 Features

We need to extract the features of the moving body which is a binary object in a video. We will describe three groups of extracted features, each group was used to categorize a group of actions. As it shown in the experiments part.
First we need to compute some global terms like area, perimeter, center of the body, global rectangle, upper rectangle (around the hands), and bottom rectangle (around the legs).

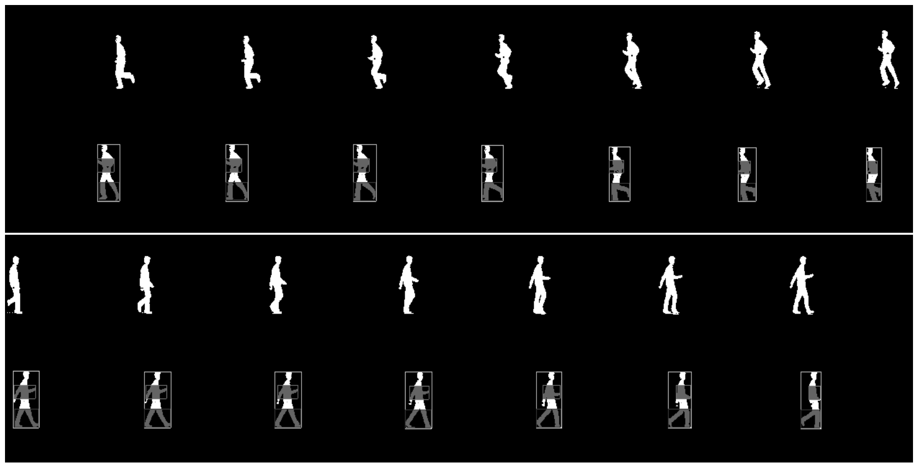


Figure 8.1: video frames with its interesting regions 'rectangles'

8.2.1 First group

This group of features are used to categorize the two global classes: the first class which describes five actions which needs to move all the body to be done and changing its place like (Walking, Jumping, Running, Skipping and Walking a Side); and the second class which contains five actions that need to move only part of the body (mostly the hands) or to move the body in the same place like (Bending, Jumping in the same place, Waving two hands, Waving one hand and Jacking).
This group consists of three features.

1. **Difference of Area**

   which are calculated on the Motion History Image of each video. This feature describes the area of the Motion History Image of the human body moving in the video. The area here is calculated by counting the white pixels in the Motion History Image which describes the movement of the body, this area is divided by the area of the body in the first frame.

$$DA = \frac{area(MHI)}{area(Frame0)}$$

2. **Center variance**

   First we calculate the Center of each human body in each frame of the video. Then we calculate the variance of these center points on the X and Y axis. Center is calculated for each frame in the binary video by:

   $$C_x = \frac{1}{A} \sum_{P \in object} P_x \text{ and } C_y = \frac{1}{A} \sum_{P \in object} P_y$$

   where A is the Area, The variance is calculated by:

   $$Variance_x = \frac{1}{n} \sqrt{\sum_{i=1}^{n} (C_i x - \overline{Cx})^2}$$

   and

   $$Variance_y = \frac{1}{n} \sqrt{\sum_{i=1}^{n} (C_i y - \overline{Cy})^2}$$

   Where $\overline{C}$ defines the mean center

3. Distance of Action

   After the calculation of the center of the object (human body) in each frame of the video. then taking the Min and the Max centers of the object in a video. We calculate the distance between the two centers (Min, Max) which describes the length of the movement. We take the distance on the X axis, because the movement in the videos is on this axis.

   $$Distance = C_{max} - C_{min}$$

8.2.2 Second Group

This group of features describes the features for actions in $ClassI$. These features have calculated on the upper rectangle (around the hands), and bottom rectangle (around the legs). The upper rectangle here defines the place of the hands which is always in the upper half part of the body, and the bottom rectangle defines the legs which is always in the third bottom part.

1. Length Difference of the global rectangle

   This takes the difference between the maximum and the minimum length (Height) of the global rectangle of the object.

   $$Diff\_H = (H_{max} - H_{min})/H_{min}$$

2. Length Variance of the global rectangle

   This calculates the length of the global rectangle of the object in each frame and calculates the variance of these lengths (Heights).

$$Variance\_H = \frac{1}{n}\sqrt{\sum_{i=0}^{n}(H_i - \overline{H}^2}$$

   where $\overline{H}$ defines the mean center.

3. Width Difference of the upper rectangle

   This feature takes the difference between the maximum and the minimum Width of the hands rectangle of the object (the upper rectangle).

$$DiffW\_upper = (Wupper_{max} - Wupper_{min})/Wupper_{min}$$

4. Width Difference of the bottom rectangle

   It takes the difference between the maximum and the minimum Width of the legs rectangle of the object (the upper rectangle).

$$DiffW\_buttom = (Wbuttom_{max} - Wbuttom_{min})/Wbuttom_{min}$$

5. Center variance on the X axis

   This feature is the same as in the first group: the variance of the center points on the X axis. Center is calculated for each frame in the binary video by:

$$C_x = \frac{1}{A}\sum_{P \in object} P_x$$

   The variance is calculated by:

$$Variance_x = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(C_i x - \overline{Cx})^2}$$

   where $\overline{C}$ defines the mean center.

8.2.3 Third group

This features will be used to categorize the run, skip, jump actions in the *classI*. It consists of two features which use the convex-hull of the human body in the video.

1. Convex Hull variance

   After finding the convex-hull of the human body in each frame of the video, We take the Area of the human body and its convex-hull in each frame. We calculate the difference between the area of the human body and the area of its convex hull in each frame and calculate the variance of these differences. The variance is calculated by:

   $$Mean = \frac{1}{N} \sum_{i \in N} area(convex_i) - area(object_i)$$

   where N = number of frames. The variance is calculated by:

   $$Variance = \frac{1}{N} \sqrt{\sum_{i \in N} (area(convex_i) - area(object_i) - Mean)^2}$$

2. Difference between Convex Hull and Object

   This feature describes the difference between the area of the human body and the area of its convex hull in each frame, and it takes the maximum and the minimum differences. It returns as result $\frac{maximum - minimum}{minimum}$.

8.3 Experiments

We aim to categorize these ten human actions: *walking, bending, jumping, jumping in the same place, running, skipping, walking a side, waving two hands, waving one hand and jacking.* We used data-base of 89 videos from [23], which are 9 videos for each action except 8 videos for the skip action. They were taken from 9 persons. Each video contains a stable background which makes it easy to segment and extract the moving body in a video as the object. In the next parts we work on the binary videos where the human body is the object in each video. It turned out to be hard to find which features are good to categorize these ten actions, because the diffusion maps uses the distances between all the examples. The distances are different in the same group of action videos. This leads to the need of finding a strategy for the categorization of the ten actions. First we categorize the ten actions as two classes: *classI* contains actions that need to move all the body like (Walking, Jumping, Running, Skipping and Walking a Side); *classII* contains actions that need to move only part of the body (mostly the hands) or to move the body in the same place like (Bending, Jumping in the same place, Waving two hands, Waving one hand and Jacking). To do this categorization we used the three features described as the first group features -(difference area, Distance of Action and Center variance on x axis)- as vector descriptor extracted from the MHI transformed image of our videos. Results showed that by using these features as input to a Diffusion Maps, it is possible to categorize the ten actions into the two classes like we described above.

Now, after the two classes were categorized well, we applied the categorization of a diffusion maps using 14 geometrical moments in 3d (2d+t) as features

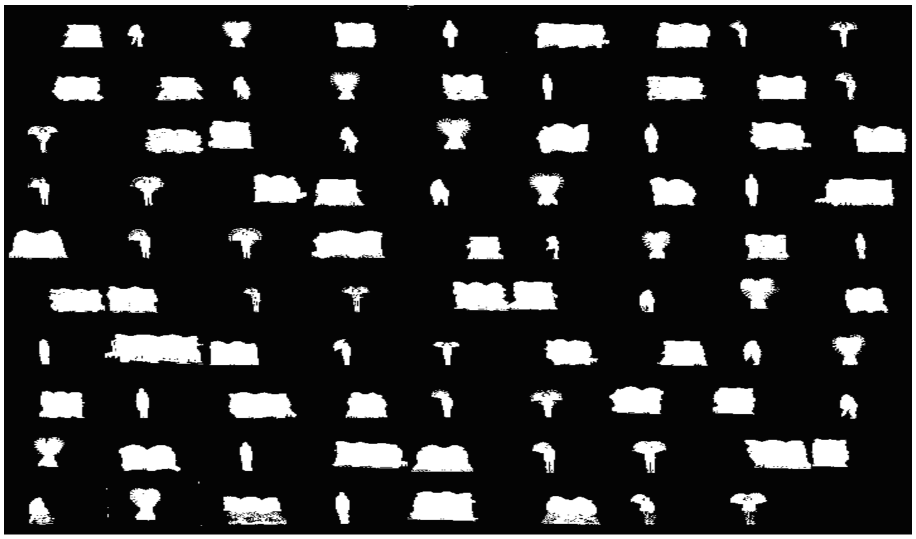


Figure 8.2: MHI for the 10 actions

descriptor to categorize the second class (*classII*) aiming to recognize what is the action in this class. The 14 moments were calculated on the MHIDT images; it gave good results.

There was only one mistake in all the videos in this class (45 videos of *classII*), After the process of the spectral clustering we can decide what is the action in this class by applying the recognition function using the Nystrom extension method. We will get then $\lambda_1, \lambda_2, \lambda_3$ as result. From the results, we found that the actions in this group can be categorized by only λ_1, as following:

λ_1	**Action**
≤ -0.33	Bend
$-0.23 < \lambda_1 \leq -0.1$	Jump in place
$-0.33 < \lambda_1 \leq -0.23$	Wave one hand
$-0.1 < \lambda_1 \leq 0.1$	Jack
$\lambda_1 \geq 0.1$	Wave two hands

For *classI* actions, we tried several features the geometrical moments, shape descriptors and 2d moments. It was difficult to categorize the actions in this class correctly. We found that the best way to categorize these actions is by separating again this class into some classes, for some action groups we get some good results; For the bad results we will show later what their action is, by using anther features. We found that using the five features (Length Difference of the global rectangle, Length Variance of the global rectangle, Width Difference of the upper rectangle, Center variance on the X axis, Width Difference of the bottom rectangle), described in the features part as the second features group,

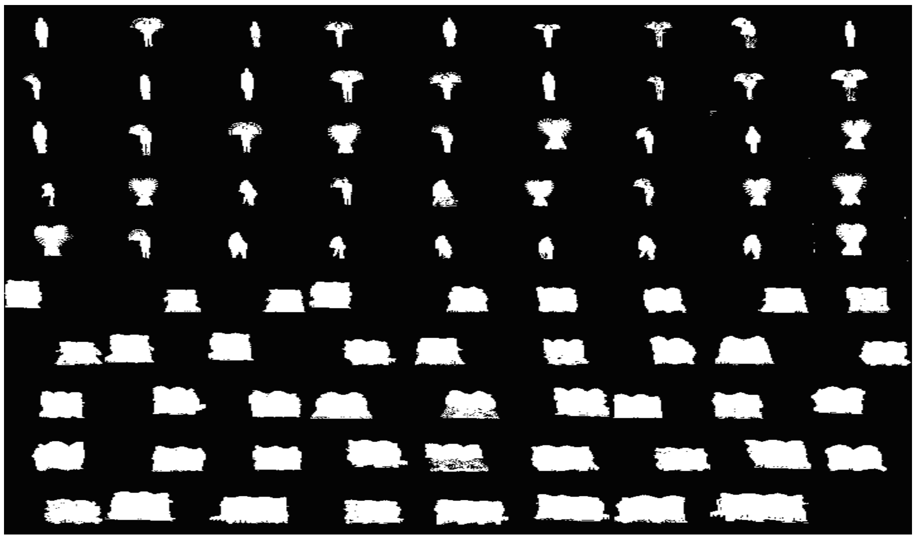


Figure 8.3: MHI of the 10 actions after first diffusion maps categorization $\sigma = 0.001$

it is possible to categorize the five action in this class into three classes using again the Diffusion Maps techniques *"spectral clustering"* with $\sigma = 0.005$.

λ_1	λ_2	λ_3	**Action**
> 0	> 0	< 0	Walk
> 0	> 0	> 0	Walk a Side
*	*otherwise*	*	Run, Skip, Jump

Finally we calculated two features, described in the features section as the third group, (variance of the (convex Hull - object), and the Difference between convex Hull and object). To decide what the action is from the third class we use here a new diffusion map to separate the three actions. We define the categorization in the recognition process by $\lambda_1, \lambda_2, \sigma = 0.365$ as the following

λ_1	λ_2	**Action**
< 0.1	> 0	Skip
< 0.1	< 0	Jump
> 0.1	*Whatever*	Run

Results were quite good here. In these three actions we have 26 examples of three actions, our approach categorized and recognized well 24 videos. However, two videos from present the skip action gave wrong answers.
Finally, our results for all the data-base videos achieved activity recognition rates above 96.6%. This demonstrates that, without any previous learning, our technique performs very well as human motion categorization method.

The categorization and recognition process flow:

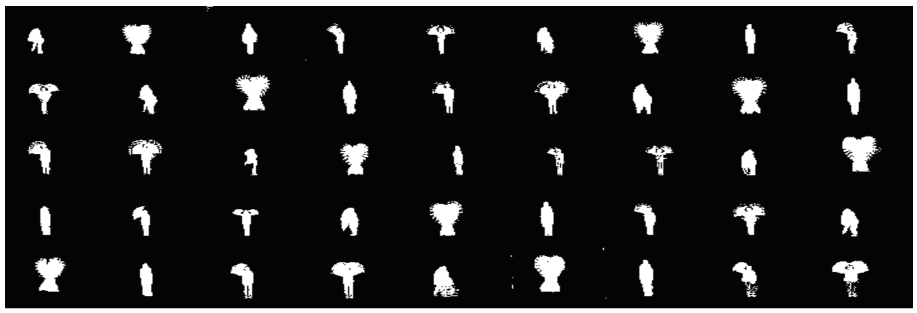


Figure 8.4: MHI of the five actions belonging to class II

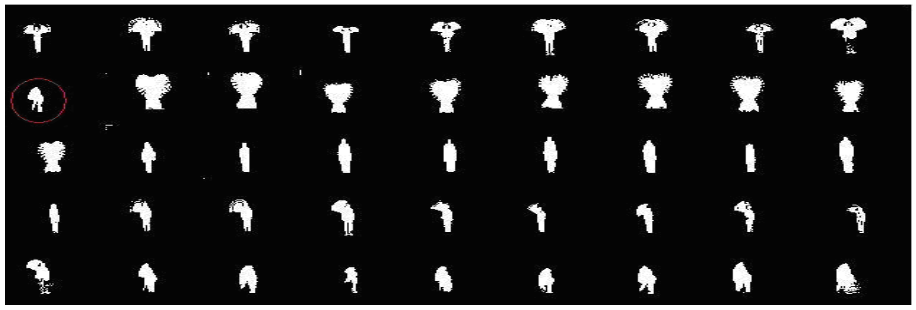


Figure 8.5: MHI of the five actions belonging to class II, the order was done by the first Eigen vector with $\sigma = 0.000045$

8.4 Conclusion and future works

A new approach based on Motion Descriptors, mass features, 3D geometrical moments and Diffusion Maps for human motion action categorization and recognition is presented. The proposed framework classifies haptic properties through the video analysis of human motion actions. Ten motion action have been tested walking, bending, jumping, jumping in the same place, running, skipping, walking a side, waving two hands, waving one hand and jacking.
We present Binary videos, each video contains a human who is doing an activity. We calculate three mass features -(difference area, Distance of Action and Center variance on x axis)- from the MHI image of the video to separate the ten actions into two classes, using the Diffusion Maps. By the Kernel method and the nystrom extension algorithm we define the two classes.
For the first class *classI*, five features -Length Difference of the global rectangle, Length Variance of the global rectangle, Width Difference of the upper rectangle, Center variance on the X axis, Width Difference of the bottom rectangle- were calculated and presented as input vectors. We categorized the five actions

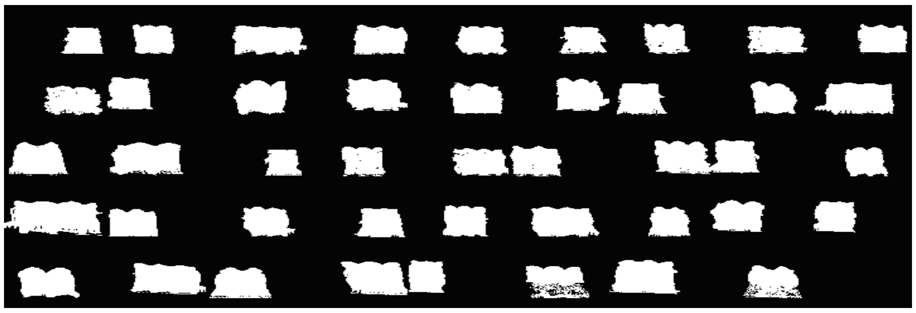


Figure 8.6: MHI of the five actions belonging to class I

in this class as three groups (First: walk, Second: walk a side, third: (run, skip, jump)). We used two more features -(variance of the (convex Hull - object), and the Difference between convex Hull and object)- to categorize the (run, skip, jump) actions. The all categorization processes here were done by using the Diffusion Maps (spectral clustering) algorithm; the recognition processes were done by using the Kernel method and Nystrom extension method.
Robust global features are extracted, based on 3D geometrical moments for the $classII$ actions, These vectors are then used to categorize the five actions classified as the second class ($classII$) by Diffusion Maps.
We tested the proposed algorithm only on our 89 videos; the results for all database videos achieved activity recognition rates above 96.6%. This demonstrates that, without any previous learning, our technique performs very well as human motion recognition methods.

There are many avenues of future work for this part, including the recognition of the human motion actions tracked by the live camera, improve the recognition system in order to reduce the recognition errors, the recognition system should correct training data online, we also plan to improve segmentation algorithm to extract the human body in real time, also it will be very important to do a virtual reality system which simulates the human motion action.

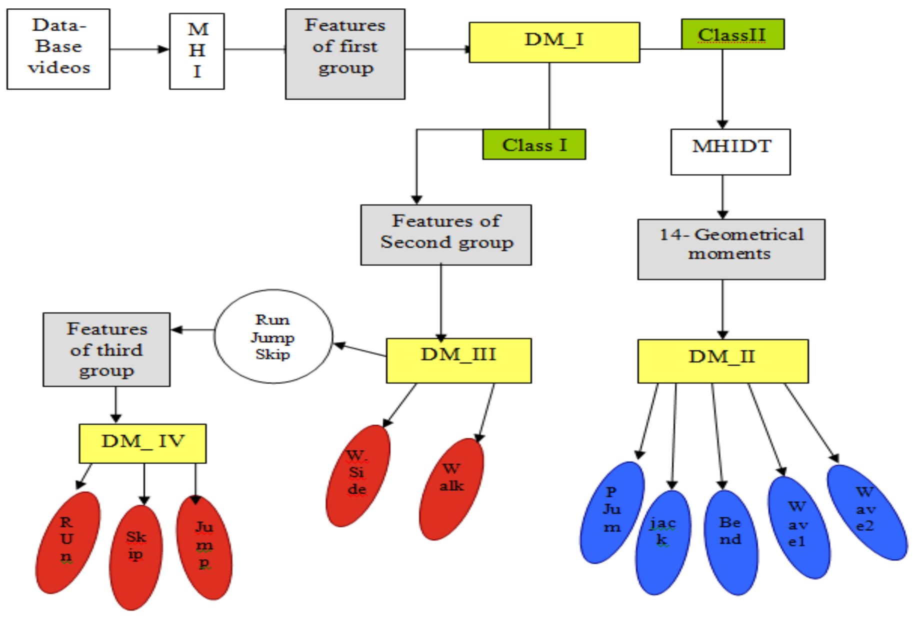


Figure 8.7: Human actions categorization process flow

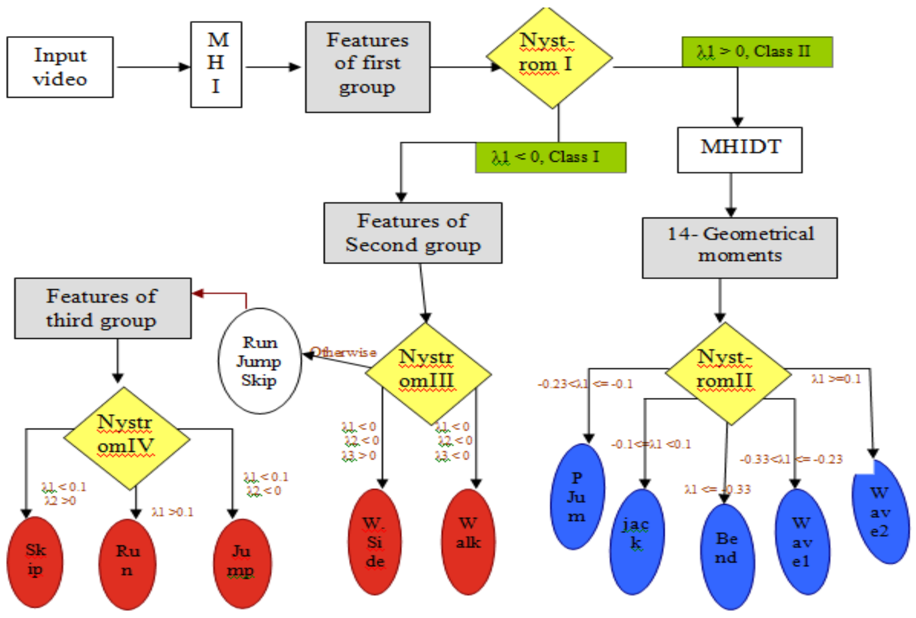


Figure 8.8: Human actions recognition process flow

Chapter 9

References

1 S.J. Lederman and R.L. Klatzky, R.L. Hand movements: A window into haptic object recognition. Cognitive Psychology, 19(3), 342-368 (1987)

2 S. Mu-Chun : A Neural Network-Based Approach to Recognizing 3D Arm Movement, Biomedical Engineering Application. Basis and Communication 15(1), 17-26 (2003)

3 X. Hou et al. Direct Appearance Models. IEEE Conf. On Computer Vision and Pattern Recognition, vol. 1, pp. 828-833 (2001)

4 T. Ogata, J.K. Tan, S. Ishikawa. High-Speed Human Motion Recognition Based on Motion History Image and an Eigenspace. IEICE Trans. On Information and Systems E89-D, 281-289 (2006)

5 S. Kumar et al. Classification of Hands Movements Using Motion Templates and Geometrical based Moments, 3, pp. 299-304 (2004)

6 M. Kenny, S.J. McKenna. An Experimental Comparison of Trajectory-Based Representation for Gesture. LNCS LNAI, vol. 2915, pp. 152-163. Springer, Heidelberg (2004)

7 M. Hammami, Y. Chahir, L.Chen et D. Zighed . Dtection des rgions de couleur de peau dans l'image. EGC03 . RSTI, Vol.17, N. 1-2-3, pp. 219-231, (2003)

8 Y. Chahir and A. Elmoataz. Skin-color detection using fuzzy clustering, IEEE-EURASIP ISCCSP . ISBN 2-808848-17-8, Marrakech, Morocco , March 13-15, (2006.)

9 Y. Boykov, and M. Jolly. Iterative graph cuts for optimal boundary and region segmentation of objects in N-D Images. Proc. IEEE 8th Int. Conf. on Computer Vision, Canada, CD-ROM, 2001.

10 G. Bradski and J. Davis, Motion Segmentation and Pose Recognition with Motion History Gradients, IEEE Workshop on Applications of Computer Vision, December 2000.

11 Youri Boykov, Vladimir Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow algorithms for Energy Minimization in Vision. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 9, pp. 1124-1137, Sept. 2004.

12 Youri Boykov, Vladimir Kolmogorov. Computing Geodesics and Minimal Surfaces via Graph Cuts.In International Conference on Computer Vision (ICCV), vol. I, pp. 26-33, Nice, France, 2003

13 L. Kotoulas and I. Andreadis. Image Analysis Using Moments, Laboratory of Electronics, Section of Electronics and Information Systems Technology, Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi 67100, Greece

14 Romer Rosales, Recognition of Human Action Using Moment-Based Features, Boston University Computer Science Technical Report BU 98-020, November 1998.

15 Areash Mokhber, Catherine Achard, Xingtai Qu and Maurice Milgram, Action Recognition with Global Factures, Laboratoire des Instruments et Systmes d'Ile de France (LISIF) Universit Pierre et Marie Curie, 4 place Jussieu, France Septembre 2005.

16 Muhamad Suzuri Hitam, Wan Nural Jawahir Hj Wan Yussof and Mustafa Mat Deris, Hybrid Zernike Moments and Color-Spatial Technique for Content-based Trademark Retrieval, Department of Computer Science University College of Science and Technology Malaysia, 2006

17 Ulrike von Luxburg, A Tutorial on Spectral Clustering, August 2006

18 Boaz Nadler, Stephane Lafon Ronald R. Coifman, Diffusion Maps, Spectral Clustering and Eigenfunctions of Fokker-Planck Operators, Department of Mathematics, Yale University, New Haven, CT 06520.

19 Pablo Arias, Gregory Randall, Guillermo Sapiro, Connecting the Out-of-Sample and Pre-Image Problems in Kernel Methods, Universidad de la Republica, Universidad de la Republica, University of Minnesota, IEEE Computer Society Conference on Computer Vision and Pattern Recognition - 18-23 June 2007

20 Matlab, Mathworks help

21 Horn, B. K. P. and Schunk, B. G. Determining optical flow. Technical Report A. I. Memo No. 572, Artificial Intelligence Laboratory (1980).

22 Ham M. Fredric and Ivica Kostanic, Principles of Neurocomputing for Science and Engineering, Mc Graw Hill International Edition (2001).

23 M.Blank, L. Gorelick, E. Sechtman, M. Irani, and R. Basri. Actions as Space-Time Shapes, IN ICCV 2005.