CLUSTERING AND VISUALIZING ACTIONS OF HUMANS AND ANIMALS USING MOTION FEATURES

by

Maryam Moslemi Naeini

B.Sc., Sharif University of Technology, Tehran, Iran, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
2007

APPROVAL

Name:	Maryam Moslemi Naeini
Degree:	Master of Science
Title of thesis:	Clustering and Visualizing Actions of Humans and Animals using Motion Features
Examining Committee:	Dr. Dr. Richard Zhang Chair
	Dr. Greg Mori, Senior Supervisor
	Dr. Mark Drew, Supervisor
	Dr. Arthur Kirkpatrick , Examiner
Date Approved:	August 23, 2007

Abstract

We propose a technique to cluster actions of humans and animals. We use domain specific motion features and employ spectral clustering on them to cluster activities. For humans, we use existing optical flow features. For animals, we cluster behaviors of a grasshopper. We track it in 3D and construct features using 3D object movement which discriminate between different classes of actions. We employ spectral clustering on the extracted features for each domain. Due to the large amount of data we use the Nystrom extension which samples from the data and computes the eigenvalues and eigenvectors of affinities between them and extends it to the eigenvectors of the full affinity matrix. We use the K-means algorithm to do the final clustering. We experimented with our method on the KTH data set and videos of one grasshopper. We create a summary visualization of our results using an extension of an existing framework.

To my husband and my parents

"There are two ways to live: you can live as if nothing is a miracle; you can live as if everything is a miracle."

— *Albert Einstein*

Acknowledgments

I am deeply indebted to my senior supervisor, Dr. Greg Mori, for his continuous support, encouragement and guidance through my research. He provided valuable insights, and a lot of help during my graduate career. This thesis would not have been possible without him.

I would like to thank my supervisor Dr. Mark Drew and my thesis examiner Dr. Arthur Kirkpatrick for being on my committee and reviewing this thesis.

I really like to thank my parents for the care and support during my studies , and last but not the least , I would like to thank my husband Hossein for his love, and care in every step that I take.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the challenging problems in the area of computer vision is recognizing actions where objects can be either people or animals. In this problem, we are given a video including one object or more and we would like to know what those objects are doing at each time.

Building a system that could do action recognition for people will save a lot of time and have many applications in video surveillance. In this application we would like to have one or more cameras monitoring people activities and generate a signal if there is an unusual behavior. To detect the unusual activity we first need to classify actions. Another application is in sport videos for the analysis of matches. We try to develop algorithms that could be used for these applications in future.

In the field of biology, recognizing actions and interactions between animals is very interesting and useful for the biologists. For analyzing animal behaviors they do experiments that require a lot of data and recording a video is one way of getting it. For these kinds of experiments they put animals in a cage or in a separate environment and monitor their activities for several days in order to find the factors that could affect their actions. In particular, they are interested in studying behavior of individual or colonies of insects under variation of environmental variables like temperature or illumination during hours of video.

Obviously processing and labeling each frame for hours of video manually is a very time consuming task either for human actions or animals. Therefore, we need an automatic system to do this task in a reasonable amount of time. The result of an recognition system is labeled frames with their actions. In these methods, the frames are labeled with their actions. This means that we mark a portion of the frames or some sequences with a label of the actions performed manually to be able to label the input frames later. This will be

hard when there are many different classes of actions and it could take a lot of time. These types of action classification methods are called supervised techniques.

The other alternative is to have a completely unsupervised algorithm that takes a sequence of frames and gives us some clusters. In the ideal case, each of them contains a separate action. This problem is more challenging and harder compared to supervised techniques because of no human supervision. Our technique is in the second category and it is completely unsupervised which is actually called *clustering*. This clustering could be used for action recognition purposes because instead of having many frames we have some clusters that contains similar actions and they could be labeled with the actions by watching some frames from each cluster.

In this thesis, we are interested in clustering actions of humans and insects. We cluster the behavior of insects called grasshopper where behavior consists of different movement of the object with different speed like walking, jumping and standing still. We present a novel method in the area of computer vision to solve the problem and cluster actions of human and insects. After clustering actions, we employ a visualization technique based on [22] to make a long video short for visualizing our results.

1.1 Approach

Our approach to solve the action clustering problem which could lead to a solution for the action recognition consists of several steps. Our method is based on the track or location of the object. First, we need to know the location of the object in each frame. Using this track we extract features either from the track or from the object pixels depending on the application type. These features are intended to discriminate between different classes of actions and be similar for the same action.

In this thesis we use two different features for the grasshopper and humans. For people we extract features based on *optical flow* or image velocity. These descriptors are vectors showing the motion of the person in all body parts in horizontal and vertical directions. For each pixel we have a flow value so pixels that move a lot have a higher flow.

Our features for the grasshopper are magnitude of differences of its 3D track in a window of frames which is a vector showing the velocity of the grasshopper within a couple of frames.

Given these features for each frame we would like to put the frames with the same action in the same cluster. So the next step is to employ a clustering algorithm that could group

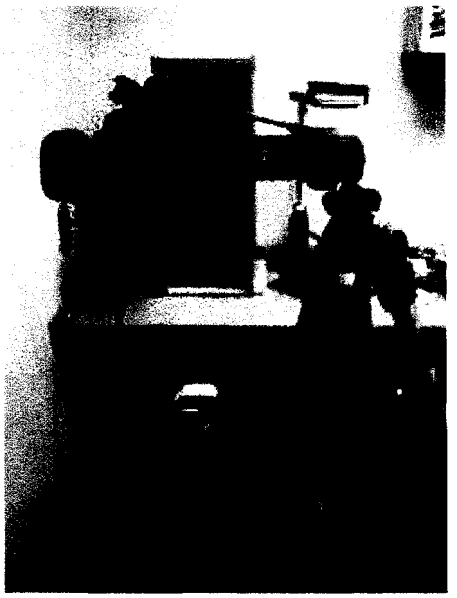


Figure 1.1: Experimental Environment for the grasshopper, two cameras are used for 3D tracking.

frames based on these features. The clustering method that we use here is called *spectral clustering*. This clustering technique uses the similarity between each pair of features and employs eigenvalues of the similarity matrix to cluster the data points. As mentioned in the introduction we are dealing with hours of video which means many frames considering a typical frame rate which is 30 frames per second. For example we use 80000 frames for one of the grasshopper experiments which is less than 44 minutes but the size of similarity matrix will be 80000 by 80000 and computing the eigenvectors of that is not efficient. To solve this problem our method employs a sampling technique called the *Nystrom extension* [21] which makes the algorithm work very efficiently. After running the clustering technique, there will be unlabeled clusters of frames which experiments show to correspond well to clusters of frames with the same actions so they could be easily labeled manually. In this section we present a brief overview of each step.

1.1.1 Motion Features

The inputs to these parts are sequences of frames. The first step is to track the object of interest because given the track we are able to extract temporal and spatial information

around the object in the image.

For each frame of human video we extract the area around the person and generate a figure centric image. This is done by using the tracker by Sabzmeydani and Mori [23]. We want to extract our features from these figure centric images that have the person in the center. The reason for using figure centric frames not the whole image is that our features are based on the relative motion of body parts and if there are any camera motions we cannot compute the features correctly.

In the case of the grasshopper, we track it in 3D. The reason for 3D tracking is to have the location of the object and its motion in 3D. In a 2D image we can only find motion in horizontal,X, and vertical direction ,Y, but we also need to find the motion along Z direction which cannot be computed using a single 2D image. Hence, we need at least two cameras to track the object in 3D. We use background subtraction as our tracking technique. Object pixels can be obtained by subtracting each frame from a background image and thresholding the difference image. Then, we create a 3D track of the insect, via 2D tracking in videos from each of the cameras that we know their parameters [6]. Figure 1.1 shows our experimental environment and indicates how we set up two cameras.

In the following we explain the motion features that we use for each class of problem.

Motion Features for Human

Given figure centric images described above, we extract motion features called optical flow. Optical flow is a vector field showing the motion of pixels. It is calculated as the motion between two image frames which are taken at times t and t + δ_t.

From the optical flow of an image we can identify moving parts of it. The reason for using this feature is that optical flow has higher values in body parts that have more motion. For example when a person walks he has more motions in his legs and when he does hand waving or hand clapping the motions are mostly around the hand of a person. Also, optical flow is invariant to appearance for example clothing and background of a person. Therefore, the features are still similar if we have persons with different clothes but performing the same action.

Figure 1.2 shows optical flow for walking, boxing and hand clapping.The optical flow values are shown as arrows. The length of arrows shows the magnitude of the optical flow. The flow is higher in the motion regions.

Estimating optical flow has applications in computer vision for recognition purposes,

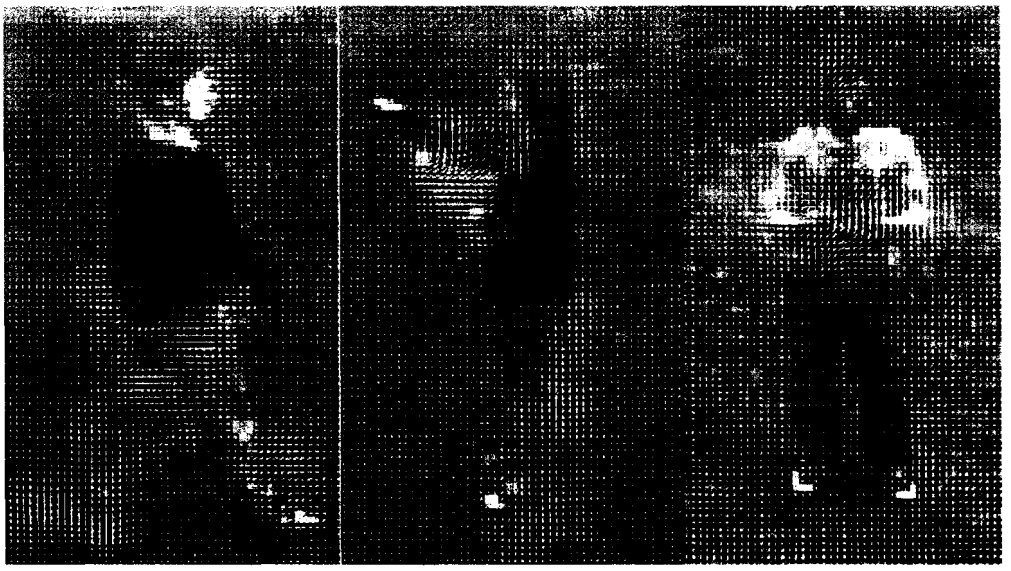


Figure 1.2: Optical flow features for different class of actions. Right to left , walking flow around the hand and the legs, boxing flow around the hands, hand clapping flow around the hands.

robotics for robot navigation and 3D reconstruction. In this thesis, we use the approach given by Lucas-Kanade [15] to compute optical flow.

Motion Features for Insect

Since we want to know when a grasshopper walks or jumps or stands still we develop features from its 3D track. Our features need to describe the movement of the object in a 3D space. Then from those features we can identify different speed and cluster similar actions to the same group.

Figure 1.3 is one frame from one camera. As it can be seen in this picture, the object is very small and it is really hard to be seen in detail. For this experiment we should have the full view of the cage in the frame to be able to see all the movements of the object inside of it. Considering all these limitations typical features usually used for human can not be extracted for the grasshopper.

Given the 3D track of the grasshopper in a long video sequence, we extract features which show the movement of the object between frames. Our goal is to develop these feature vectors to get high values when the insect jumps and low values for standing still.

Figure 1.3: Grasshopper study environment. Box around the object is the tracker result.

We compute feature vectors that are magnitude of differences between 3D coordinates. Our experiments show that these features discriminate between different actions and lead to correct clustering.

1.1.2 Clustering Motion Features

After constructing motion features for each domain for each video segment in the case of the grasshopper and for each frame in the case of the human subject we have a vector that describes the corresponding action. We would like these vectors be different for different actions. Now we need an automatic technique to group these features into clusters which in the ideal case each cluster should include one or similar actions.

There are different clustering methods and classifier the technique that we use here is called spectral clustering. Spectral clustering is a clustering method used for image segmentation, and recently action recognition and clustering [19,35]. In spectral clustering a similarity matrix is constructed between each pair of the data. Then, the dominant eigenvectors of this matrix are used to cluster the data.

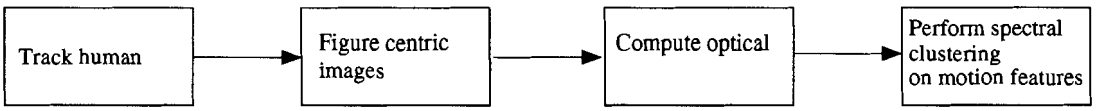


Figure 1.4: Overveiw of the techniques used for people.

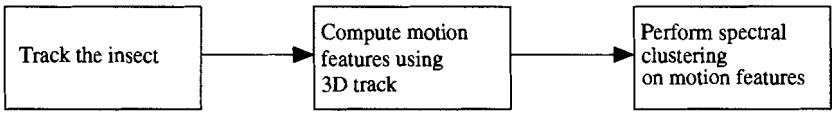


Figure 1.5: Overview of the technique used for the insect.

Here, we define a similarity between groups of frames based upon a motion cue. For humans, we use the optical flow vectors as the data points. We employ a measure called *normalized correlation* between normalized optical flows of each frame. As mentioned optical flow can be seen as a vector field so at each pixel we have a vector. Computing *normalized correlation* is similar to the *scalar product* or *dot product* of two vectors so when the vectors are aligned perfectly the dot product will get its maximum value.

For the grasshopper, we use magnitude of differences between 3D position which shows motion of the insect as features and the data points. For computing the similarity we use the exponential function of negative euclidean distance between them which is a typical measure for computing the affinities in spectral clustering.

When we are dealing with hours of video, the amount of data will be very large and computing the eigenvectors and eigenvalues of the similarity matrix will be computationally expensive. To avoid this problem, we employ the Nystrom extension [10, 21], a sampling technique that can be used in computing eigenvectors for spectral clustering. Our contribution is to develop a fast and accurate action clustering in long video sequences with use of spectral clustering, and a particular method of sampling, Nystrom extension, to cope with rare actions. Figures 1.4 and 1.5 show the overview of our technique for human and insect.

1.1.3 Visualizing Results

Watching hours of video either for analyzing animal activity or for monitoring people actions needs a lot of time. It makes people bored watching videos in which nothing happens. In this work we use the existing visualization framework of Rav-Acha et al. [22] that summarizes hours of video and makes a very short video called a *synopsis*. Depending on the application

this shorter version of the video could include all the frames and objects in the original video or not, but more importantly it is much shorter and more interesting to be watched. We use their technique for visualizing the activities in each cluster separately but in a very shorter time. To summarize either the original video of grasshopper or merging different sequences of human action we put multiple instances of the object in one frame using the 2D track information of the object and minimizing the overlap cost between objects. By applying these techniques we not only have a much shorter video that includes all the actions but also we could evaluate the correctness and performance of action clustering results just by looking at the summarized version.

1.2 Contributions

The contributions of this thesis are introducing a new algorithm for action clustering that is employed for both humans and animals. Our technique is based on extracting previously developed motion features for humans. For insects we give novel features based on the movement of the animal.

Given features we employ spectral clustering to cluster these features which correspond to frames of video. We also use the previously developed sampling extension called Nystrom method for spectral clustering but for the first time for action clustering. The Nystrom method makes it more efficient and improves the performance by computing the eigenvectors of similarities between samples and extending it to the eigenvectors of the full similarity matrix. This is an important issue in processing hours of video either for surveillance purposes or biological studies where both the inputs are hours of video.

The other important benefit is that our technique is completely unsupervised which means that nothing is done manually to process the input and there is no need for manually labeling the data for training like supervised techniques. There are other unsupervised techniques but our method handles large data as well as being unsupervised.

The last contribution is that we employ the result of our action clustering method in an existing visualization framework. We visualize the result of each cluster in a separate video which is much shorter than the input video and include all the information. Doing this helps us to observe all the activities for a long video in a very short version also we could evaluate how good our technique is by looking at the actions from each cluster in a shorter amount of time.

1.3 Outline

The rest of the thesis is organized as follows: we explain the previous work mostly in area of action recognition in Chapter 2. In Chapter 3, the preprocessing and tracking that is required for each class of problem will be discussed. Chapter 4 describes the action features in details and spectral clustering technique. In Chapter 5, we will discuss the visualization technique used to make a short video of clustered action sequences. Chapter 6 presents the results of our technique for both humans and the grasshopper. Finally we conclude this thesis and discuss future works in Chapter 7.

Chapter 2

Previous Work

The problem of action recognition has always been one of the main challenges in the area of computer vision. Many efforts have been done in this area to develop and improve recognition algorithms. Gavrila [11] analyzed a lot of algorithms in human tracking and action recognition. The survey paper by Forsyth et al. [9] is about existing tracking algorithms motion synthesis for human. Moeslund and Granum [17] reviewed methods in human tracking, pose estimation and action recognition.

In this section, we mainly present some of the previous works that are discussed. Since our method includes different phases and preprocessing, the previous work in each of areas need to be mentioned separately. The topics we refer to as related works can be outlined as:

- Tracking
- Action Recognition
- Spectral clustering
- Visualization

In the following sections we will present some of the existing algorithms.

2.1 Tracking

As mentioned before, first we need to find the object in the image. The object track provides useful information that can be used in recognizing actions. The object tracking problem

has received a lot of attention in literature. There is a survey paper by Lepetit and Fua [14] on tracking literature.

There are advanced approaches in tracking animals like using particle filters by Khan et al. [12,13]. They use both appearance and location of the object in particle filters to track bees. In particle filtering each particle represents the probabilistic location of the object. These particles are weighted according to their probability to be the object. In each time a particle is chosen randomly and based on its weight. Then it is moved using the motion model and weighted again using its new appearance. Since employing particle filters need a motion model and the grasshopper movement is very unpredictable and with very different speeds we cannot use particle filters to track it.

Another set of the methods that is simple and widely used for tracking objects is background subtraction [27,32]. The basic idea is to detect foreground by taking the difference between current frame and background. If there are slight changes in background or there is some camera motion the technique will fail. In Toyoma et al. [27], some background maintenance techniques are presented which could improve the performance of background subtraction. This paper proposes technique to update the background at different level which are:

- Pixel Level: The pixel-level algorithm makes a probabilistic prediction value for every pixel at each time. It uses the past values up to time t to predict the value of the pixel at time t. Then the difference between a predicted value and its real value is computed and if it is large the pixel is considered as a foreground. Hence, if there are frequently changing pixels this value is low and it is high for sudden changes that correspond to the foreground.

- Region Level: The region level considers relationships between pixels and works in such a way that instead of isolated pixels there will be regions and segments of the object.

- Frame Level: It is for adapting to sudden changes in the background. Multiple models were used for the background. The system switches between models if necessary. For example if there is a sudden change in the lighting the background model will be changed for all pixels.

Another approach is to use a mixture of Gaussian distribution to model the values of

a particular pixel as a mixture of Gaussians. This model helps to take care of multiple background objects that are changing a lot. For example leaves of a tree around a building that are moving but they are parts of background. Stauffer and Grimson in [26] use this probabilistic model for each pixel value x at time t. A Gaussian density function is shown in Eq. 2.1. μ and Σ are the mean and co-variance of the Gaussian. The weighted mixture of these functions is presented in Eq. 2.2.

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|} e^{\frac{-1}{2}(X_t-\mu_t)^T\Sigma^{-1}(X_t-\mu_t)} \tag{2.1}$$

$$P(X_t) = \sum_{i=1}^{k} \omega_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{2.2}$$

In this equation K is the number of Gaussian distributions. ω is the weight function. Each of this Gaussian function represent a model at each pixel and is weighted in the whole model.

In Wren et. al. [32] a Gaussian average of the frames is taken and set as the background. Which means at each time a new Gaussian mean or average is computed and used to find the foreground from background. The average μ_t is :

$$\mu_t = \alpha I_t + (1-\alpha)\mu_{t-1} \tag{2.3}$$

I_t is the current frame and the trade off between stability and quick update defines α. The standard deviation of Gaussian function, σ, is used to find the foreground. Pixels that satisfy Eq. 2.4 are classified as the foreground.

$$|I - \mu_t| > k\sigma_t \tag{2.4}$$

Since we are using a static background for the grasshopper and we want to have a robust background subtraction we employ this technique to track the grasshopper which could take care of slight changes in the background.

2.2 Action Recognition

Recognizing actions of humans or animals is one of the challenging areas of computer vision. There have been a lot of works in this area specially for people due to applications in

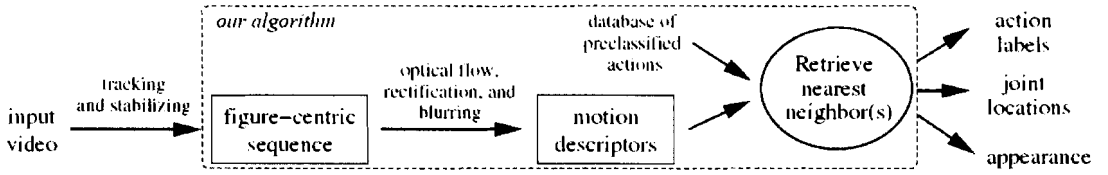


Figure 2.1: Data flow for Efros et al. algorithm. Starting with a stabilized figure-centric motion sequence, they compute the spatio-temporal motion descriptor centered at each frame. The descriptors are then matched to a database of pre-classified actions using the k-nearest-neighbor framework. The retrieved matches can be used to obtain the correct classification label, as well as other associated information. (Figure and caption from [8] ©2003 IEEE, by permission)

surveillance, sports and human computer interaction. In this section, we will discuss some of these works.

Some of the approaches for action recognition use optical flow or image motion. There are different algorithms to compute optical flow for images in a sequence. In a paper by Barron et al. [2] different algorithms for computing the optical flow are compared.

Efros et al. [8], use optical flow as the motion descriptor for each frame of the sequence. We used the idea of this work in this thesis. The flow of their algorithm is shown in Figure 2.1. They use figure centric images to eliminate camera motion and compute the optical flow for each of these figures. The optical flow is separated into four different channels corresponding to motion in four different directions. Figure 2.2 shows each of the channels for a football player. Channels are blurred to remove the noise. After obtaining these motion descriptors, the similarity between them is computed using normalized correlation. To have similarity between motions that are similar but occur at different rates, the similarity is blurred with a blurry identity matrix which is sum of rotated identity matrices as shown in Figure 2.3. A typical frame to frame similarity matrix before and after blurring is shown in Figure 2.3. Given this matrix, nearest neighbor technique is used to find frames with similar actions. They also use these motion descriptors to do action synthesis.

Another approach using optical flow is given by Zhu et al [36]. They present a technique to recognize tennis player actions in a video. Histograms of optical flow and Support Vector Machines (SVM) classifier [29] were used to classify different actions.

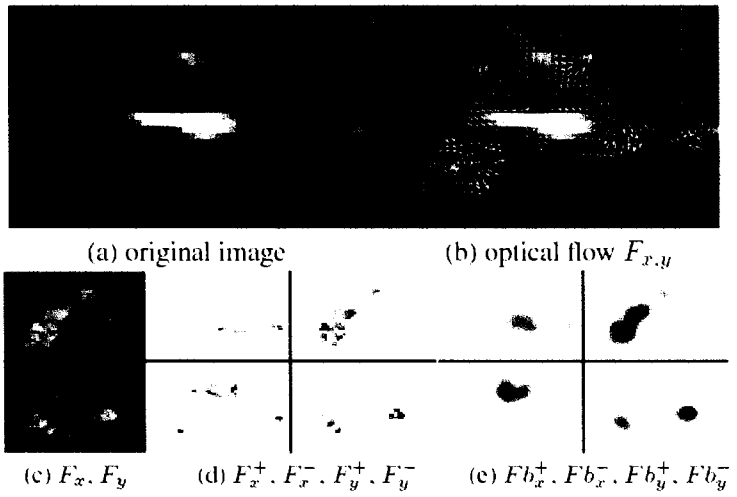


Figure 2.2: Constructing the motion descriptor (a) original Image, (b) Optical Flow, (c) Separating x and y component, (d) half wave rectifation of each component to produce 4 separate channels, (e) Final Blurry motion channels. (Figure and caption from [8] ©2003 IEEE, by permission)

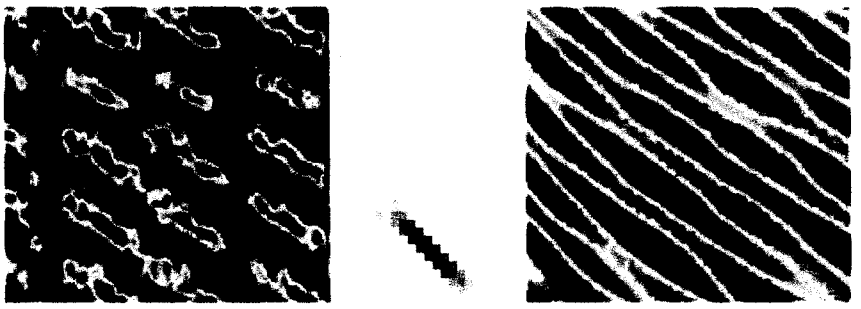


Figure 2.3: (a) A typical frame-to-frame similarity matrix S_{ff} for running, (b) the Blurry I kernel K (not shown to scale) used for aggregating temporal information within the similarity matrix, (c) the resulting motion-to-motion similarity matrix S. (Figure and caption from [8] ©2003 IEEE, by permission)

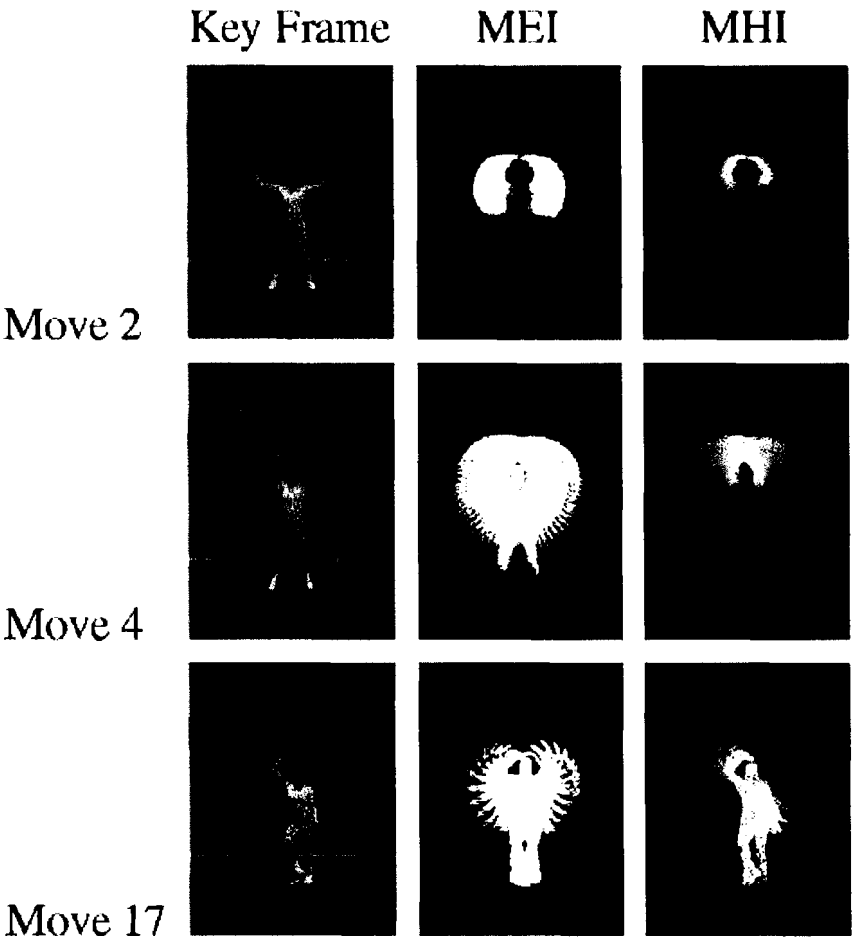


Figure 2.4: Comparison of MEI and MHI. Under an MEI description moves 4 and 17 are easily confused; under the MHI, moves 2 and 4 are similar. Because the global shape descriptions are weighted by the pixel values, having both images yields more discrimination power. (Figure and caption from [5] ©2001 IEEE, by permission)

Figure 2.5: Space-time shapes of jumping-jack, walking and running actions. (Figure and caption from [4] ©2005 IEEE, by permission)

Bobick and Davis in [5] develop an algorithm based on templates. Two different templates are computed that are based on the silhouette of the object using background subtracted images. For matching the object movement to a template, a motion energy image (MEI) and a motion history image is computed (MHI). The MEI is the sum of background subtracted images over time, and in the MHI intensities of pixels are function of motion during time. Hence, brighter pixels in MHI have moved recently. Figure 2.4 shows the MHI and MEI for various moves of person. They try to find the best matches between input frames and templates using MEI and MHI.

Blank et al. [4] present a technique that is based on computing a space-time shape of the object. The space time shape includes a lot of spatial information at each time step. The space time shape is shown in Figure 2.5. They solve an equation using space time shape of a person and extract saliency features and orientation of body parts. Then, the authors employ nearest neighbor technique using these features to evaluate these features

Wang et al. [31] present an algorithm for clustering actions in still images using deformable shape matching and spectral clustering. To avoid computing a very large similarity matrix, a fast pruning based on a descriptor called a shape context [18] is done to keep only a set of promising candidates. Then, deformable shape matching cost is employed for computing the affinities for spectral clustering. This is done by sampling points from the edges and finding an optimal assignment matching cost between them. They test their approach on sport data sets.

Dollar et al. [7] and Belongie et al. [3] analyze actions by first tracking and then computing spatio-temporal patch features. The authors use an approach based on detecting

interest points by filtering in space time domain. Interest points around a local maxima of the response are extracted and called cuboids. After extracting a local space time volume, different types of features such as normalized pixel values, brightness gradient, or optical flow is computed as the features. K-means clustering is used to create library of cuboid prototypes. Therefore, it is expected that similar cuboids should be on the same prototype. Histograms of cuboid types used as descriptors and for comparing the test cuboid to the prototypes. This work is similar to its previous work by Shuldt et al. [24] but it uses slightly different features.

Other works on automatically analyzing the behavior of animals include Balch et al. [1], who have developed methods for tracking multiple ants, and suggest the use of Hidden Markov Models for analyzing their behaviors.

Zhong et al. [35] build a co-occurance matrix over vector quantized spatial motion feature and video segments. They use spectral clustering to cluster actions. In their case, the co-occurance matrix is sparse, and eigensolving is efficient.

2.3 Spectral Clustering

Clustering has always been a difficult task when the data dimension is high or it is very large and has a complex distribution. Recently spectral clustering has received a lot of attention and used in different areas like image segmentation [10,25] and action recognition [4,19,31,35]. This method is based on constructing a similarity or affinity matrix and using dominant eigenvalues of this matrix to cluster the data points. Verma and Meila in [30] compare different spectral clustering approaches. Ng et al. in [20], present a simple spectral clustering algorithm. The affinities in their method is exponential function of negative of euclidean distance between points. A diagonal matrix constructed which has the sum of rows of affinity matrix in its diagonal. These two matrices are multiplied. Then the K dominant eigenvectors of the result matrix are used as the embedding coordinates in a k dimensional space and the K-means clustering used to cluster these points instead of original points.

Shi and Malik in [25] use the spectral clustering for image segmentation. The image segmentation problem is phrased as a weighted graph partitioning problem. The weights shows the similarity between pixels. Then, normalized cut is used to partition this graph. Finding the min cut of a graph G means partitioning the vertices of graph into two sets such that the sum of the weight of the edges between these two sets is minimized. Normalized

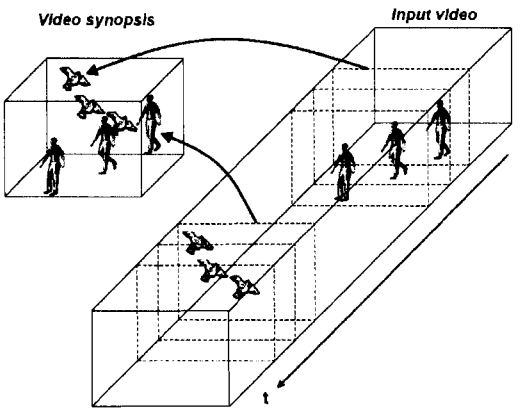


Figure 2.6: The input video shows a walking person, and after a period of inactivity displays a flying bird. A compact video synopsis can be produced by playing the bird and the person simultaneously. (Figure and caption from [22] ©2006 IEEE, by permission)

cut is similar but instead of minimizing the sum of edge weight the cut cost is computed by finding the fraction of total edge connections to all vertices in the graph.

Minimizing the normalized cut is NP complete but Shi and Malik [25] show that this problem can be relaxed to a generalized eigenvalue problem. They used their algorithm for image segmentation. The second eigenvector of the similarity matrix is recursively used to bipartition the points which in this case are image pixels.

Fowlkes et al. [10] used a technique called Nystrom extension for solving the eigenvalue problem by sampling from points and finding an approximation for the eigenvectors. They suggest how to apply this extension to the Normalized Cut. This technique can be very useful when we want to process large data and it will highly improve the performance. They use their algorithm for image segmentation. We will use it for action clustering.

Zelnik-Manor and Perona [33] discuss some important issues in spectral clustering like scale, σ, in the Gaussian function usually used for similarity computation. We use their approach for computing the similarity between features used for the grasshopper.

2.4 Visualization

In this work, we are dealing with processing hours of video. Watching them either for video surveillance, sports or biology purposes need spending a lot of analyst time. Therefore, if

we visualize and summarize what happened in the long video in a very short version we save a lot of time.

We use a technique similar to work in [22] to visualize classified actions. Rav-Acha et al. in [22] present a very interesting technique that can help summarizing a long video to a very short version. Figure 2.6 shows schematic space time volume for the original and synopsis which the shortened sequence. The original video includes a walking person and flying birds at different times but the synopsis video has both of these simultaneously.

Summarizing is done by dividing video into segments and minimizing a cost function for a time shift M and a subset selection of segments B. The cost function includes:

- E_o : Occlusion cost between objects pixels in two frames. The occlusion cost is computed between each pair of segments.
- E_a : Activity loss which is the difference between the number of active pixels or object pixels in the original and synopsis video.
- E_l: The length cost of the synopsis video.

The cost function is shown in Eq. (2.5) [22].

$$E(M, B) = E_a + \alpha E_o + \beta E_l \tag{2.5}$$

They use simulated annealing to minimize the objective function and obtain the segments in the shortened video. They also presented how to apply this technique for lossless synopsis video. A lossless synopsis is a shorter version that includes all the video segments in the original video and its main application is in the video surveillance where we want to observe all the activities. One of the result frame of a lossless video synopsis is shown in Figure 2.7. We use this idea and employ it to visualize our action clustering results, and generate synopsis for insect and people datasets.

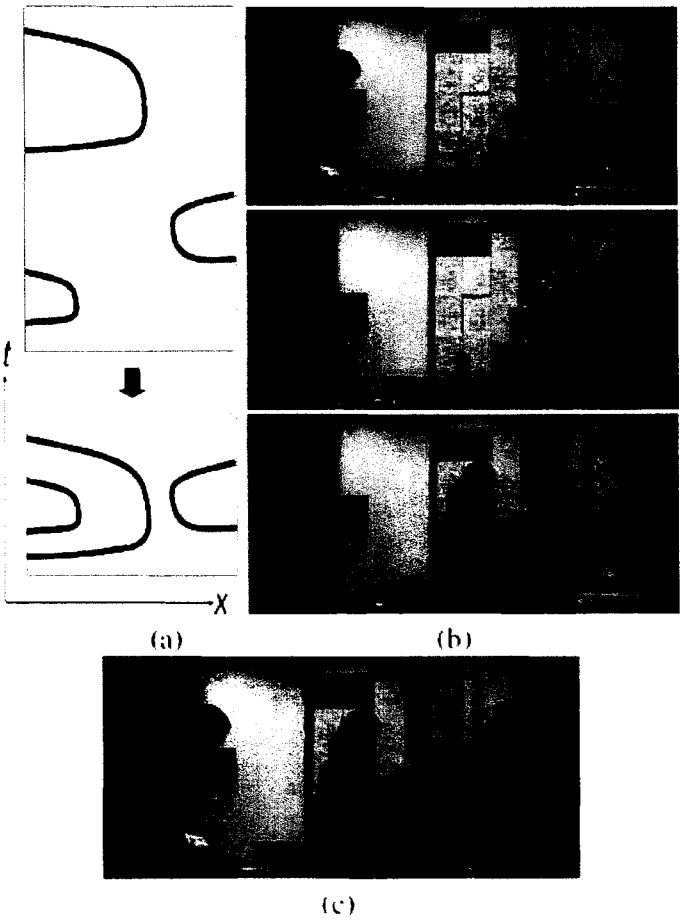


Figure 2.7: An example when a short synopsis can describe a longer sequence with no loss of activity. Three objects can be time shifted to play simultaneously. (a) The schematic space-time diagram of the original video (top) and the video synopsis (bottom). (b) Three frames from original video. (c) One frame from the synopsis video. (Figure and caption from [22] ©2006 IEEE, by permission)

Chapter 3

Camera Calibration and Tracking

The first step is to track the object since its track gives us a lot of spatial and temporal information that we can use for extracting motion features. The input to this step is a sequence of frames and the output is either the figure centric frames for humans or the 3D track for the insect.

For humans we simply run a tracker by Sabzmeydani and Mori [23] which is based on edge features and Adaboost learning. Since the data set we use has only one person at each frame, tracking is easy and the track output is reliable. After that, we extract the figure centric image which is a window around each person and no preprocessing of the input for humans is necessary.

For the grasshopper we are interested in the 3D track of the object to be able to say where it is at each time step and compute its movement.In this section we will discuss these steps for the grasshopper since the 3D tracking of the grasshopper need some preprocessing. Our approach is based on separate 2D tracking and then generating the 3D track from two 2D tracks. To do this, some preprocessing is necessary to find the camera parameters. In the following section we will discuss the preprocessing and 3D tracking of the grasshopper.

3.1 Camera Calibration

We need to find the 3D track for the insect. Using one camera enables us to extract only the 2D track of the object. We need at least two cameras to be able find the 3D track and to be able to project between 2D and 3D we need to find their parameters. The process of finding a camera optical and geometrical parameters is called *calibration.*

Each camera has a set of optical or intrinsic parameters. Camera intrinsic parameters are:

- Focal length: The focal length, f_x, f_y.
- Principal point: The center of the image, c_x, c_y.
- Skew coefficient: The cosine of angle between x and y axis, α.
- Distortions: The radial and tangential distortions.

Each camera also has a set of geometrical or extrinsic parameters which are used to transform between a known world reference frame and unknown camera reference frame. Camera extrinsic parameters are:

- Rotation: R rotation matrix between camera frame and world reference frame.
- Translation: T translation matrix between camera frame and world reference frame.

The relation between a 2D point in an image and its 3D coordinate in world reference frame is given by Equation 3.1 :

$$sm = PM \tag{3.1}$$

Where m=[x,y,1] and M=[X,Y,Z,1] are the homogeneous coordinates for the corresponding 2D point in the image and its 3D world coordinate. P is the projection matrix $P = K[R|T]$ where K is 3 by 3 matrix containing intrinsic parameters of a camera:

$$\begin{pmatrix} f_x & \alpha f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

R and T are the rotation and translation between the world and camera reference frame.

To find the camera parameters we use Bouguet's camera calibration toolbox in Matlab. In this technique, a calibration pattern, for example a checkerboard which is shown in 3.1 is used. The checkerboard is printed and attached onto a planar surface. We have to show this pattern from different angles to the camera. The details of the proper angle between the image plane and the pattern for obtaining good calibration results is analyzed in [34].

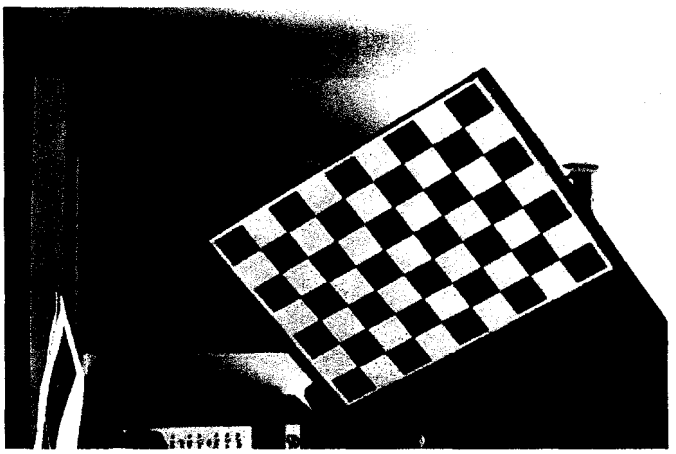


Figure 3.1: Calibration Pattern.

The whole pattern should be seen in both cameras. For each view we have to click on four corners manually. After having enough images of the checkerboard the toolbox can find the parameters in K. Details of solving the method and the number of planes required is discussed in [34]. For each shot the toolbox computes the corners of checker board by searching the image and using camera parameters. Minimization of mean square error between them used to compute more accurate intrinsic parameters.

Given two 2D tracks which is the output of background subtraction technique in section 3.2 we find the 3D track by employing the *triangulation* procedure which exists in the same toolbox. Equation 3.1 does not have an inverse which means that given a 2D coordinate we cannot find the 3D coordinate that corresponds to it. That is because each point in 2D correspond to a line in 3D which means infinite number of points. Therefore, the 3D position (X,Y,Z) of a point P_{3D}, can be reconstructed from the projection of P_{3D} on the image planes of at least two cameras, given the relative position and orientation of them.

So we need at least two calibrated cameras. Calibrating two cameras and finding the relative translation and rotation between them is called stereo calibration. To do this we first calibrate each of the cameras separately by the process explained on top then the toolbox is able to recompute the camera parameters by doing stereo calibration. The important issue here is that having the cameras synchronized. Because the views used for calibrating cameras should be the same to be able to do stereo calibration. For synchronization we use a digital clock in the beginning of recording our video.

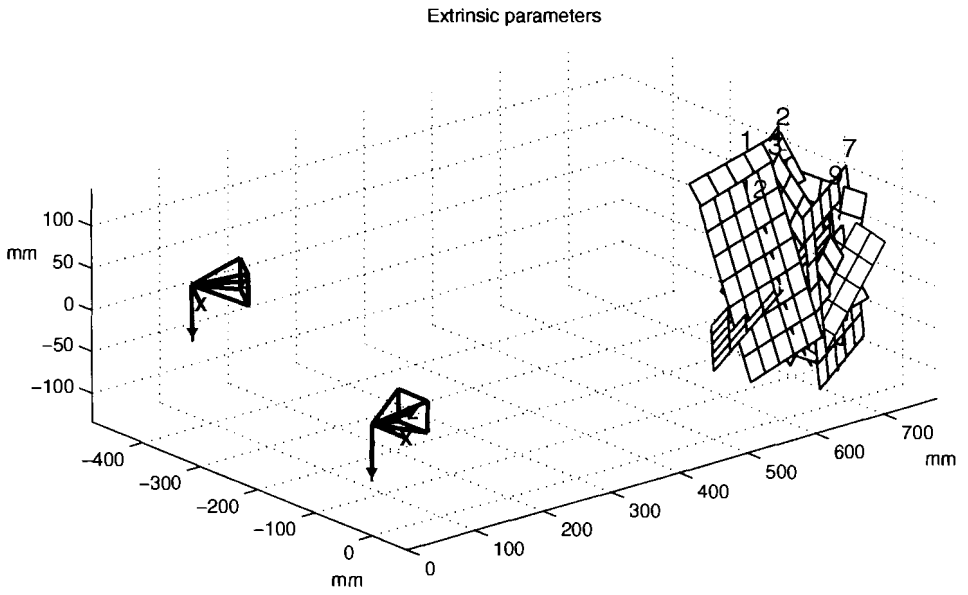


Figure 3.2: Extrinsic Parameters of cameras including different views of the checkerboard used for calibration.

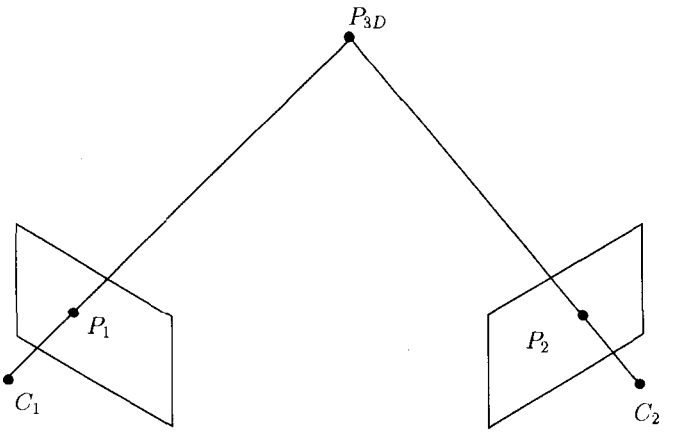


Figure 3.3: Projecting from 2D to 3D.

We set up two cameras as shown in Figure 1.1 and calibrate them by employing the toolbox. We use 15 different views of the checkerboard and show it to the cameras under different orientations. The toolbox finds the focal length, principle point, skew, distortion also the translation and rotation between the cameras. The extrinsic parameters given by the toolbox is shown Figure 3.2.

Figure 3.3 shows how we back-project from two 2D points to a 3D point. In this figure C_1 and C_2 are the camera principle points. P_1 and P_2 are image of one P_{3D} point. We can find the 3D coordinate by taking the intersections of two lines that pass through the 2D coordinates and camera principle points. Due to error in calibration the lines usually do not intersect. In that case, the 3D location will be the point between lines that has minimum distance from both of the lines. By doing stereo triangulation we can compute the 3D location of the object given two 2D tracks.

3.2 Stereo Tracking

Tracking this insect is difficult due to its very small size and its color changes as it is walking. In addition, the insect makes occasional jumps which are so fast that sometimes it is very hard to be seen, for example a 50 cm jump only occurs at 10 frames considering 30 frames per second. To overcome these difficulties, we used a fixed painted background and image differencing to detect the object, instead of tracking algorithm based on color histogram or motion models of the target, which tend to oversmooth the jumps.

Figure 3.4: Difference image before smoothing.

Figure 3.5: Difference image after smoothing.

We employ a background subtraction technique to track the object. We smooth the difference image I_d using a Gaussian filter. We take the pixel of maximum sum of red, green and blue values as center of the location of the insect as shown in Eq. 3.2.

$$x_o = \arg(\max_{x \in I_d}(I_d^r(x) + I_d^g(x) + I_d^b(x))) \tag{3.2}$$

In this equation, x_o is the grasshopper center, $I_d(x)$ is the difference image, and $I_d^r(x)$, $I_d^g(x)$ and $I_d^b(x)$ are the red, green, and blue values of each pixel. The difference image is shown before and after smoothing in Figure 3.4 and Figure 3.5. The noise is mainly because of presence of slight changes in the background, for example in the border of the cage. Our background image is set to be the average of all the frames up to time t to make the algorithm more robust to slight changes in illumination or other variations in background image [32]. Employing these techniques we track the object in each of the cameras separately and obtain the 3D track by performing the stereo triangulation procedure explained above.

Chapter 4

Action Clustering

In this chapter we will discuss the core of our action clustering algorithm. We start with either figure centric images around the person or the track of the grasshopper. We are interested in clustering different actions for both cases. The first step is to construct features that could discriminate between different classes of actions. We will describe our novel features for grasshopper and the existing optical flow vector field features for human in separate sections. Given these features for frames we need a clustering algorithm to put the same actions into a same cluster. In this thesis we discuss how to employ the spectral clustering technique using the Nystrom extension on the features to cluster different actions.

4.1 Action Features

Given the figure centric image or track of the object we first need to extract features from them that could describe actions in the frames. Since features depend on the domain of the problem they are different for video of humans or a video of one grasshopper. We are using two different motion features, one for each class of problem. In the following subsections, the details of features for each case are discussed.

4.1.1 Motion Features for Human

In this section, we introduce the motion features that we use for human. Our motion features are based on the optical flow or image velocity. As mentioned before, optical flow is a descriptor that shows the motion of the image. There are different algorithms to compute

optical flow which were discussed in Barron et al. [2]. The motivation behind optical flow is that it shows the motion regions for a human body which is different for different actions, also it is invariant to appearance. So a person action is described by the optical flow features.

In this thesis, we use the algorithm by Lucas and Kanade [15,16] to compute optical flow for each frame. The Lucas Kanade method is one of the popular methods for computing optical flow using derivatives in space and time. The first assumption is the image brightness constancy constraint which means that the appearance of the object does not change as it moves. Hence, we have [28] :

$$\frac{dI(x,y,t)}{dt} = 0 \tag{4.1}$$

Therefore, for every point (x, y) in image I that moves by (dx, dy), we can write:

$$\frac{dI(x,y,t)}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t}\frac{dt}{dt} = 0 \tag{4.2}$$

$$\frac{\partial I}{\partial x} = I_x, \frac{\partial I}{\partial y} = I_y, \frac{\partial I}{\partial t} = I_t \tag{4.3}$$

$$\frac{dx}{dt} = F_x, \frac{dy}{dt} = F_y \tag{4.4}$$

$$I_x F_x + I_y F_y + I_t = 0 \tag{4.5}$$

$$\nabla I \cdot F = -I_t, F = (F_x, F_y) \tag{4.6}$$

$$\tag{4.7}$$

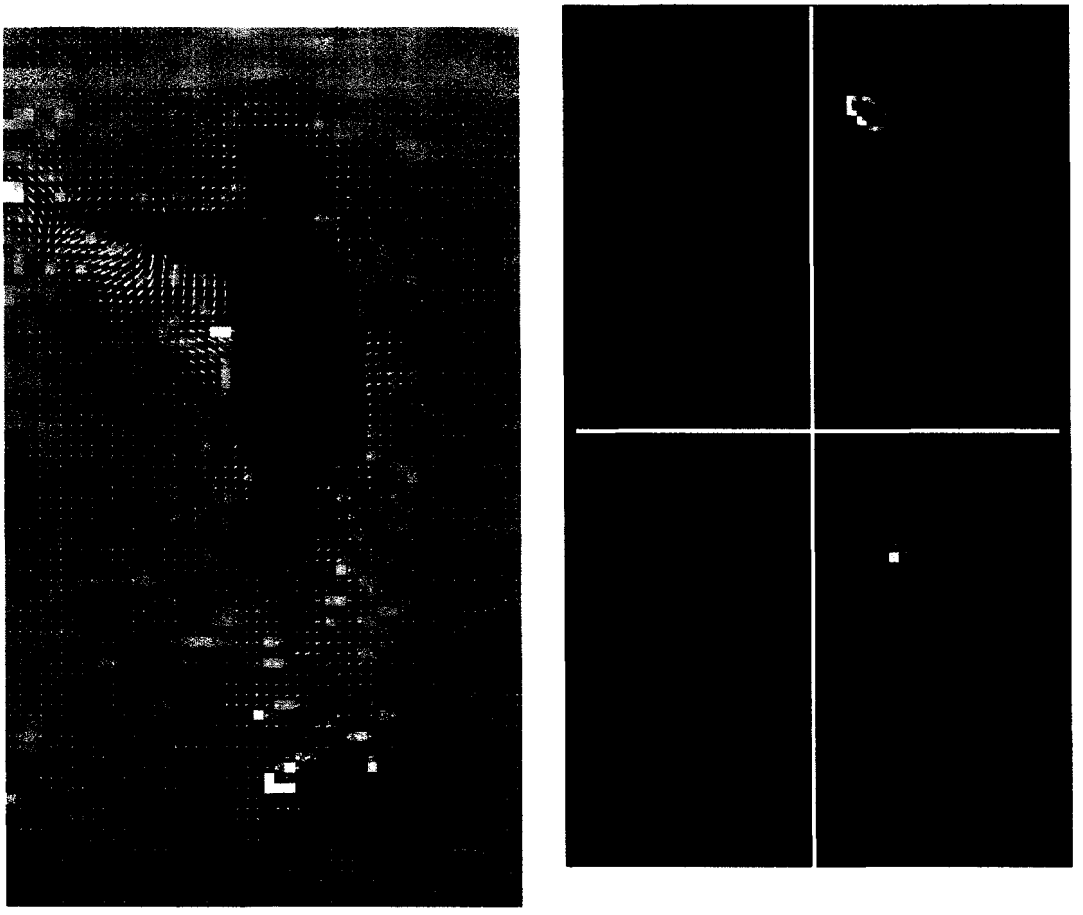
F_x and F_y are the x and y component of optical flow and ∇ is the partial derivatives of I with respect to x and y. Assuming that the flow is constant in a small window, the solution to this problem can be found by solving a linear system of equations.

Since we want to represent the motion in four possible directions(up, down, right, left) which are non-negative, we half wave rectify x and y channels of optical flow to four non-negative channels. This is same as Efros et al. [8].

$$F_x = F_x^+ - F_x^- \tag{4.8}$$

$$F_y = F_y^+ - F_y^- \tag{4.9}$$

Next, we blur each of these channels using a Gaussian filter. The blurring is done to remove the noise in the flow computation for example location of the arm while boxing



(a) (b) Blurred flow channels. F_x^+,F_x^-,F_y^+,F_y^- (clockwise)

Figure 4.1: Optical flow for a boxing person.(a)Original image,(b)Four blurred optical flow channels showing motion in four directions. Flow values changes from high to low in range of colors from red to blue. There is a high flow around the right hand of the person in F_x^- and F_y^-.

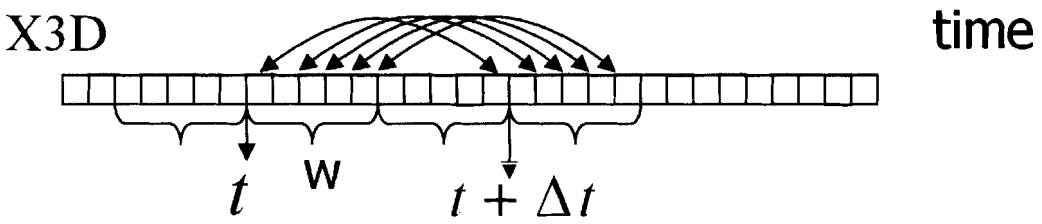


Figure 4.2: Feature vectors based on movement of the object.

might vary. Then, we normalize these vectors for each image to be able to compare them later. For normalization we normalize each flow channel separately so the sum of flow values for each flow channel over the image is one. The flow for a boxing person from KTH data set is shown in Figure 4.1.

4.1.2 Motion Features for Grasshopper

We track the object in 3D and specify the location of it at each frame. The next and more challenging step is to cluster different actions of the insect such as jumping, walking and standing still using its movements between frames. Although the tracker always points to the object, the location information is noisy. This noise is more when the object is not moving which makes the clustering task more difficult.

We define a set of motion features based upon this tracker output which we will use to describe grasshopper tracks. The motion features will be clustered using spectral clustering. Obtaining a good motion feature is a critical task that impacts the quality of clustering. The word 'good' means that the feature should be as different as possible between the actions which are in two separate classes, and as similar as possible between actions within a same class. The classes of actions we are considering for grasshopper is jumping, walking, and standing still.

Constructing the motion feature is a crucial part and since we are using only the 3D position we smooth that using a Gaussian filter Eq. 4.10 to remove the noise in the tracker output.

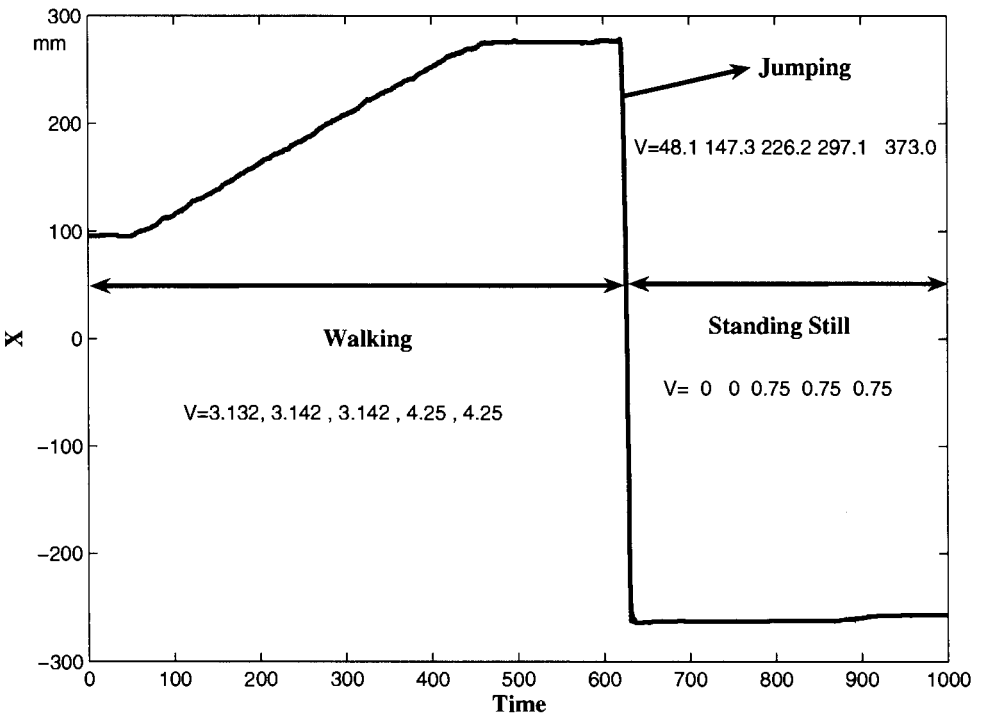


Figure 4.3: 1D track of the object and a sample feature vector for each action during the track.

$$x_s(t) = x(t) \star G(t) \tag{4.10}$$

$$G(x) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} e^{\frac{-x^2}{2\sigma^2}} \tag{4.11}$$

Then for each non-overlapping window of size W of 3D position of the object we compute the difference between $x_s(t)$ (location of grasshopper in 3D at time t) and $x_s(t+\delta_t)$ for each of the frames in this window. This feature is illustrated in Figure 4.2. So our feature vector V_t for window of size W of 3D coordinates sequence in time will be:

$$V_t = (|x_s(t) - x_s(t+\delta_t)|, |x_s(t+1) - x_s(t+1+\delta_t)|, \ldots, |x_s(t+W) - x_s(t+W+\delta_t)|) \tag{4.12}$$

We will perform clustering on these W-dimensional feature vectors. The motivation for

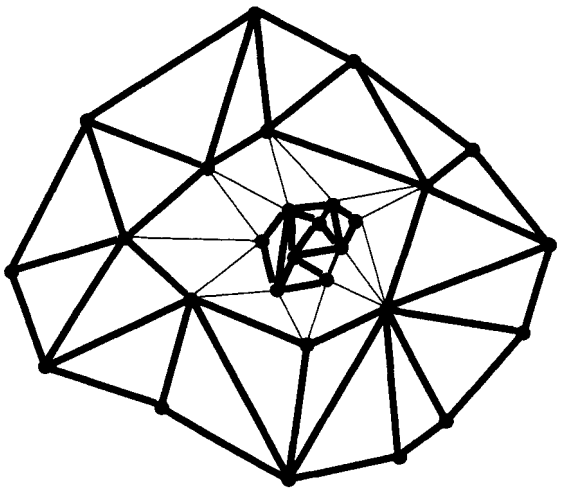


Figure 4.4: In spectral clustering data points are the nodes and affinities between them are weighted edges of a complete graph. The thickness of the lines shows the edges weight. (for clarity some of the edges are removed)

employing these features instead of the gradient of x_s is to eliminate noises in the tracker by using δ_t frames instead of 1 frame used for gradient computation. So when the grasshopper is standing still but the tracker is noisy, this feature is designed to smooth the gradient computation. The 1D track of the object for different actions and a sample feature for that action is shown in Figure 4.3.

4.2 Spectral Clustering

Until now we have some features that correspond to one or more frames. We want to cluster these features into classes that each of them in ideal case should represent an action. There are different approaches for clustering, we have chosen spectral clustering. Spectral clustering is a clustering method that uses the eigenvalues and eigenvectors of a similarity matrix between data points to cluster them. In spectral clustering, an affinity or similarity matrix W that indicates the similarity between each pair of data points is constructed. Therefore, in two dimensional case, data points and similarity between them can be illustrated as a weighted complete graph as shown in 4.4. W_{ij} of this matrix stores the similarity between nodes i and j. In our work, nodes i and j are motion features corresponding to frames. The data is clustered by analyzing the eigenvalues and eigenvectors of this matrix. We compute

eigenvectors of W and use the dominant eigenvectors. Then, we cluster data points using k-means in the embedding space given by these eigenvectors.

Computing the right affinities plays a key role in clustering result. The measure used to compute affinities is very important and should be different depending on the problem domain. In ideal case, W is strictly block diagonal under a permutation of its rows and columns. It means W is nonzero between similar actions and zero between different classes of actions. In the following sections, we will describe the affinities that we use for our clustering techniques.

4.2.1 Computing Similarity Matrix

In this section, we discuss different measures that we use for human and grasshopper to compute the affinity or similarity between motion features.

Similarity for Human Features

As suggested in [8], we use normalized correlation to compute the similarity between four channels of optical flow that correspond to motion in four different directions. Since each of these channels is normalized the correlation between them is a normalized correlation. Consider frames i and j of sequences A and B. We have four channels of flow : $a_1^i, a_2^i, a_3^i, a_4^i$, $b_1^j, b_2^j, b_3^j, b_4^j$ for frame i in sequence A and frame j in sequence B. There are blurred and normalized flow channels in four directions. To normalize them we treat each channel of flow for each n by m frame as n by m vector and divide the flow of each pixel by magnitude of that. Then, We compute the affinities between them using [8]:

$$W(i,j) = \sum_{t \in T} \sum_{c=1}^{4} \sum_{x,y \in I} a_c^{i+t}(x,y) b_c^{j+t}(x,y) \tag{4.13}$$

This is used since optical flow is a vector field and this is similar to computing the dot product of vectors from each image which is reasonable measure for computing the similarity between two vectors. T and I are the temporal and spatial windows. The last two sums can be calculated using the formula given by Efros et al. [8] in Eq. 4.14 where columns of A_i and B_i includes the flows for channel i for each frame which are reshaped as vectors. For example if sequence A has k frames of size n by m the size of matrix A_i for each i is nm by k.

$$W_{ff} = A_1^T B_1 + A_2^T B_2 + A_3^T B_3 + A_4^T B_4 \tag{4.14}$$

This formula only uses the spatial window and it is equal to Eq. 4.15. Which is part of the whole correlation formula.

$$W_{ff} = \sum_{c=1}^{4} \sum_{x,y \in I} a_c^{i+t}(x,y) b_c^{j+t}(x,y) \tag{4.15}$$

We have to sum W_{ff} over a temporal window by taking the convolution of W_{ff} and a blurry identity Kernel I_t to compute the final motion to motion similarity [8]. The reason for this blurry I kernel is to have higher similarity between motions that are similar but occur at slightly different rate. The affinities for our human data set before and after blurring for different action classes are shown in Figure 4.5.

Similarity for Grasshopper Features

After constructing the features in Section 3.2, we compute the distance d_{ij} between features i and j using Euclidean distance. For computing the weight we use negative of this distance and Eq. 4.16.

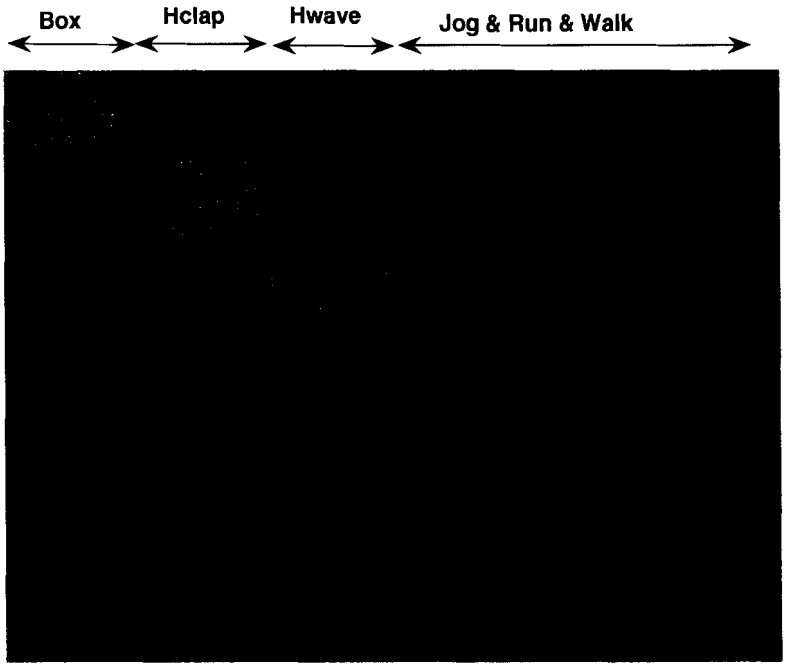
$$W_{i,j} = exp(-\frac{d_{ij}^2}{\sigma_i \sigma_j}) \tag{4.16}$$

We apply local scaling or local σ, instead of a fixed *sigma* [33]. This means that each point has its own σ and we use that to compute the affinity between that point and all other points instead of using a fixed *sigma*. The reason for not using the simple Gaussian function which has a fixed σ is that the distances between clusters are not the same. For example, if we have a tight cluster within a background cluster and use a constant scale, it leads to weights that may not describe the real similarity between features.

This problem is illustrated in Figure 4.6. It is shown that using a local σ instead of a fixed one can lead to better affinities between data points.

In this method, the scaling factor σ is a function of distance between nodes. Our choice for σ is :

$$\sigma_i = d(V_i, V_k) \tag{4.17}$$



(a) Unblurred frame to frame affinity.

(b) Blurred motion to motion affinity with blurry I kernel.

Figure 4.5: Affinities between frames using normalized correlation for all six classes of actions in KTH data set. Similarity changes from high to low in range of colors from red to blue. The similarity between walking and jogging and running is high.

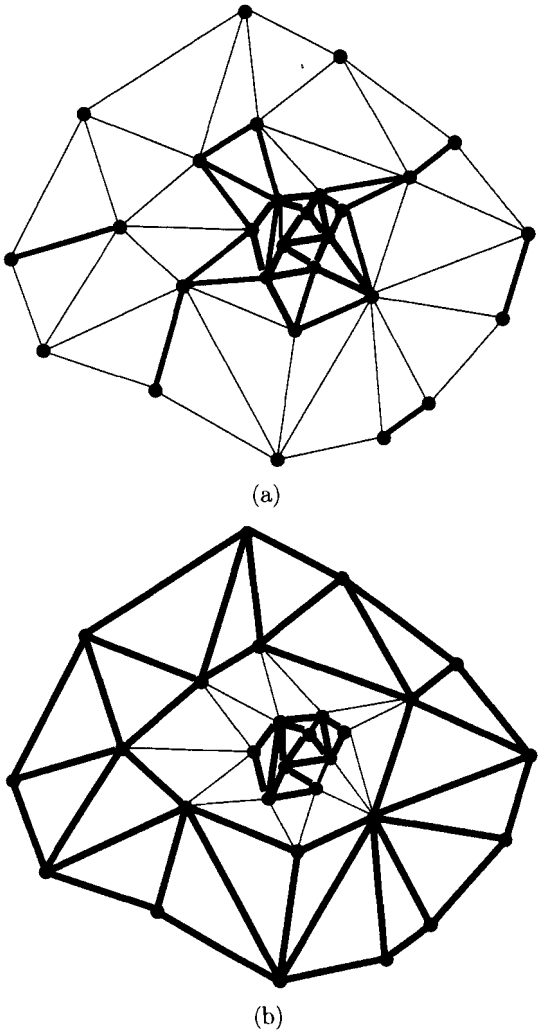


Figure 4.6: Affinities between points using (a) Fixed σ, and (b) Local σ. Thickness of lines corresponds to the magnitude of affinity.(Similar to [33])

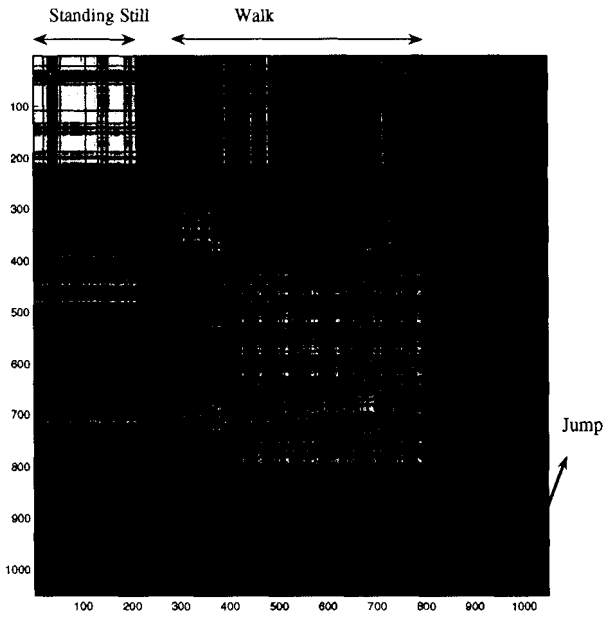


Figure 4.7: Reordered affinity matrix for grasshopper jumping, walking and standing still.

where V_k is the k_{th} neighbor of node i. k in this formula should not be very large or small. If it is large, weights between all the points become large. On the other hand, most of the nodes get a low similarity if k is small. In our experiments, k is set to 10.

The reordered affinity matrix for different class of actions has been shown in Figure 4.7. As it is clear from this picture, the number of jumping frames are much lower than standing still and walking frames so jumping is considered a rare action.

4.2.2 Spectral Nystrom

When dealing with large amount of data, computing eigenvalues and eigenvectors of a large matrix is an expensive task. For our application, there will be thousands of nodes; so, constructing, storing, and computing the eigenvectors of the matrix W will be intractable. To overcome this limitation, we apply the Nystrom extension [10, 21] which provides a method for extrapolating eigenvectors computed on a portion of W to the entire matrix. To the best of our knowledge there is no probabilistic guarantee on the accuarcy of the eigenvectors and eigenvalues given by the Nystrom method but experiments show that it

works for our data sets and for the clustering. Following the notation in Fowlkes et al. [10], given an N by N affinity matrix W,

$$W = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \tag{4.18}$$

where A is an n by n sub-matrix of W containing a set of randomly chosen sample points. If $n << N$, eigenvectors U of A can be computed efficiently, and then extended as $\bar{U}$ to the entire matrix W by:

$$\bar{U} = \begin{bmatrix} U \\ B^T U \Lambda^{-1} \end{bmatrix} \tag{4.19}$$

where Λ is the diagonal matrix of eigenvalues of A.

An important point is that if rare activities exist, and are not randomly chosen in matrix A, the extended eigenvectors given by Eq. 4.19 will not be accurate.

For the grasshopper as mentioned the number of jumping frames is much smaller than the number of standing still and walking frames. Therefore, we augment the set of random data samples with a fixed number r of data points which are jump features. Clustering experimental results show that augmenting the Nystrom samples with jump samples does not effect the Nystorm extension in eigenvector computation. These points are chosen based upon affinities in B, finding samples which are furthest away from the originally randomly chosen samples. In our experiments, we found the results to be insensitive to the setting of this parameter r.

We then compute eigenvectors and eigenvalues for this augmented matrix $\hat{A}$, and use the Nystrom extension to extend these eigenvectors to the entire matrix W. Finally, we perform k-means clustering on the resulting embedding coordinates.

We summarize our action clustering algorithm for humans and grasshopper in the following tables. The summaries for both methods are similar but for humans it needs fewer steps since we do not have rare activities for human data set.

Using these algorithms we did action clustering for humans and the grasshopper. We will use the previously developed visualization technique to visualize the results in a shorter version that includes all the actions.

Table 4.1: Action clustering algorithm for humans.

1. Construct the spectral graph using features in section 4.1.1.

2. Sample from the nodes randomly.

3. Compute affinities A, the between samples matrix using Eq. 4.13.

4. Compute affinities B between samples and rest of the nodes matrix using Eq. 4.13.

5. Compute the eigenvalues of affinities using one shot technique in [10].

6. Use the K largest eigenvectors $E = [E_1 E_2 ... E_k]$.

7. Cluster rows of matrix E which are the embedding coordinate in K-dimensional embedding space using K-means algorithm.

Table 4.2: Action clustering algorithm for a grasshopper.

1. Construct the spectral graph using features in Section 4.1.2.

2. Sample from the nodes randomly.

3. Augment these samples with the $r = 4$ furthest nodes from these samples.

4. Compute distance between samples $D^A_{nsamp \times nsamp}$ using L_2 distance.

5. Compute distance between samples and rest of the nodes $D^B_{nsamp \times nrest}$ using L_2 distance.

6. Sort rows of D^A matrix and choose the j_{th} column as σ_A compute affinities A, between samples matrix using Eq. 4.16.

7. Sort columns of D^B matrix and choose the j_{th} row as σ_B compute affinities B, between samples and rest of the nodes Matrix using Eq. 4.16.

8. Compute the eigenvalues of affinities using one shot technique in [10].

9. Use the K largest eigenvectors $E = [E_1 E_2 ... E_k]$.

10. Cluster rows of matrix E which are the embedding coordinate in K-dimensional embedding space using K-means algorithm.

Chapter 5

Visualization

In the previous chapters, we developed a novel method to cluster different actions of humans and animals. Now, we want to find a way to show the classified actions of the object to the user. Since the original video is very long, we need to summarize it into a short video to present classified actions in a reasonable amount of time. Also, we want to separate different action classes in the output video to make the clustering results visible. In this chapter, we develop a method to visualize our clustering results in a short video. We first present the overview of our method; then, we describe steps of our algorithm in details.

5.1 Overview

Given the original video of the object, we want to make a synopsis video by moving different parts of video in time and put them together as a single video. In this case, we have several copies of the same object in our visualization. Figure 5.1 illustrates the formation of the synopsis video. This approach is based on [22].

Let V be the original video, $V(x, y, t)$ is the value of pixel located at position (x, y) in the frame at time t. We first classify the action happening at each frame using our action clustering method. We extract the video segments as a set of consecutive frames belonging to the same action class. In the next step, we reorganize the segments and move them in time to make a shorter video while keeping similar actions together. Finally, we generate the synopsis video, $S(x, y, t)$, using these segments.

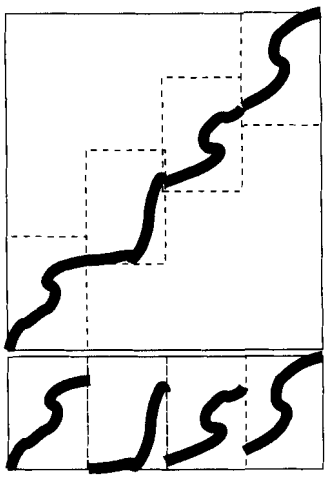


Figure 5.1: Overview of synopsis approach. Horizontal axis is the position and vertical axis is the time. This could be 1D track of one object and in the shorter version on bottom we have multiple instances of this object from different video segments of the original video on top. (Similar to [22])

5.2 Video Segmentation

In this section, we describe how we produce video segments from the original video $V(x, y, t)$ and clustering results. In video segmentation, we break V into smaller video sequences. We should define the appropriate segmentation of video V. Let $c(t)$ be the cluster label corresponding to the action of object performed at time t. An appropriate segmentation is a set $A = \{a_1, \ldots, a_n\}$ of segments. Let $T(a_i)$ be the time segment a_i starts. We are interested in building A such that all frames in the segment a_i belong to the same cluster.

We start by traversing all frames in the video sequence and crop the sequence of frames with the same action label. In particular, we start by $A = \{a_1\}$ and $T(a_1) = 1$ and do the following: We pick a frame at time t in V and verify if $c(t) = c(t-1)$. If so, we proceed to the next frame $t + 1$. Otherwise, we add a new segment a_i to A with $T(a_i) = t$.

In real applications, some objects may continue doing the same action for several minutes. In this case, the generated segment is too long. As we will describe our repositioning technique in the next section, long segments limit the length of synopsis video more than smaller segments. This is due to the fact that smaller segments can be rearranged with more freedom, and we can have more video segments at each time instance. To limit the video segment length, we define l_{max} to be the maximum segment size, and maintain

$l_i \leq l_{max}, 1 \leq i \leq n$ during the segmentation where l_i is the length of segment a_i.

On the other hand, in some cases the number of frames in a segment is less than a minimum segment size, l_{min}, since they are very short and sometimes labels are not right and caused by the noise. So, we merge the segment with the previous or the next segment with the following constraint, if the segments whose length is less than l_{min} and its neighbors are not rare actions we could apply merging to them. The action is called rare if we know that the object has such a rare behavior or if there is a cluster which is much smaller than other clusters. We take care of that action and its cluster by not merging it to its neighboring segment. For example, for visualizing grasshopper results, we can merge two segments if they are sequences of walking and standing still. However, we can not merge jumping and walking or standing still since jumping is a rare action. We find the jumping cluster by finding the cluster that has minimum size since jumping is very rare. For visualizing human videos, we do not have any rare action.

5.3 Segment Repositioning

In the next step, we want to move each video segment in time to achieve two objectives: (i) the segments have the least overlap, (ii) the final video is as short as possible. We also maintain all video segments in the synopsis video. Given a set of video segments, A, our goal is to find mapping T' where $T'(a_i)$ is the time segment a_i appears in the synopsis video S.

To minimize the overlap between video segments, we first define the occlusion between two segments i and j based on [22] as :

$$C(i,j) = \sum_{x,y,t \in S} X_i(x,y,t).X_j(x,y,t) \tag{5.1}$$

where $X_i(x,y,t)$ is 1 if there is an object at pixel (x,y) at time t of synopsis video in segment i. The total occlusion cost for mapping T' defined as [22]:

$$C(T') = \sum_{1 \leq i,j \leq n, i \neq j} C(i,j) \tag{5.2}$$

Note that $C(T) = 0$ because there are no video segments with a common frame. Therefore, $C(i,j) = 0$ for all pair of i and j, $i \neq j$. Now, our objective is to find T' that minimizes $C(T')$. We fix the length of S to l. In particular, we do the following:

We add segments one by one at random to the synopsis video. For each segment a_i being added, we set the value of $T'(a_i)$: For all values of $T'(a_i)$ between 1 and l we compute $C(T')$ and we pick the one making $C(T')$ minimum. In other words, the we try all the different time shifts for segment a_i in the synopsis video to obtain the least occlusion with previously added segmnets. We fix the value of $T'(a_i)$ and proceed with adding the next segment to S.

As mentioned before, we want to present same actions together. Therefore, we do the video segmentation separately for each action and create a separate synopsis video.

5.4 Generating Output

After finding the optimal T', we generate output frames one by one. We first find a set of segments, R, that are rendered at time t. Next, for each segment in R we find the object pixels area using the track of the object. We copy the object pixels from V to S.

Now we describe how to find R for each frame and copy objects into generated frame. For each frame at time t, $1 \leq t \leq l$, R is obtained by finding segments which satisfy:

$$T'(a_i) \leq t < T'(a_i) + l_i. \tag{5.3}$$

Next, we generate frame t of the synopsis video. At the beginning, it only consists of the empty background. Then, for each $a_i \in R$, we copy the object pixels from V to S. We proceed to frame $t + 1$ and do the same.

5.5 Experimental Results

In this section, we present the visualization results of our algorithm. We ran the algorithm for both humans and grasshopper. Since the KTH data set is not consistent in backgrounds and camera position and there are a lot of scale and lighting variations, we could not produce a good output. We ran our algorithm over 45 minutes of a single grasshopper video and produced a 2-minute synopsis video.

We ran the synopsis for each cluster separately and generated one video for each cluster one for walking, one for standing still, and one for jumping. We present a sample output frame in Figure 5.2. As you can see in the figure, 11 grasshoppers were rendered into the

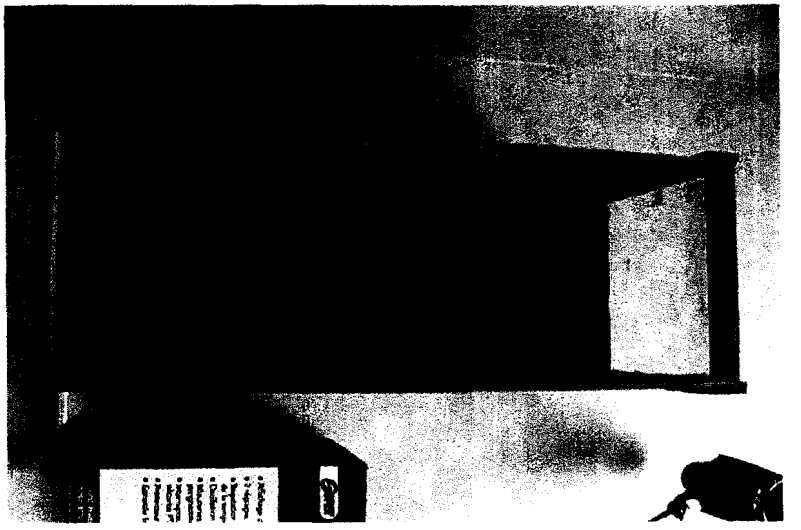


Figure 5.2: A grasshopper synopsis video frame when it walked up the wall. 11 grasshoppers in the figure are the one grasshopper in the original video at different times.

same frame although there is only one grasshopper in the original video. These are the copies of one grasshopper in different times when it walk up the wall of the cage.

Chapter 6

Experiments

In this chapter we present our experiment results for humans and the grasshopper. In the first section we give the results for humans. Next, we present results for the grasshopper.

6.1 Humans

In this section we evaluate our experimental results for clustering human actions. We use the KTH data set. This data set includes 6 different actions performed by 25 different persons 4 times changing different variables. There are a lot of variations in its videos in scales, clothing, and lighting. This six actions are boxing, hand clapping, hand waving, jogging, walking and running. One frame from each action is shown in Figure 6.1

Due to the huge size of the data set we experimented our method on 764 sequences, and choose 10 frames from each sequence. Therefore, the total number frames is 7640. In Figure 6.2 the performance of our algorithm versus different number of clusters is shown. Although we are considering only four distinct actions by putting together walking, jogging and running we did experiments with more number of clusters than 4. The reason for that is the fact that there are always variations in the data which leads to have better performance using more number of clusters than the action classes. For example all the walking, jogging and running in different directions are put in separate clusters. Therefore having more clusters improves the performance. For manual labeling of the clusters, watching some of the frames for 5 to 10 clusters including similar actions is much easier than watching all the frames of the video and by increasing the number of clusters by a small number we get better clusters which makes the labellings task easier.

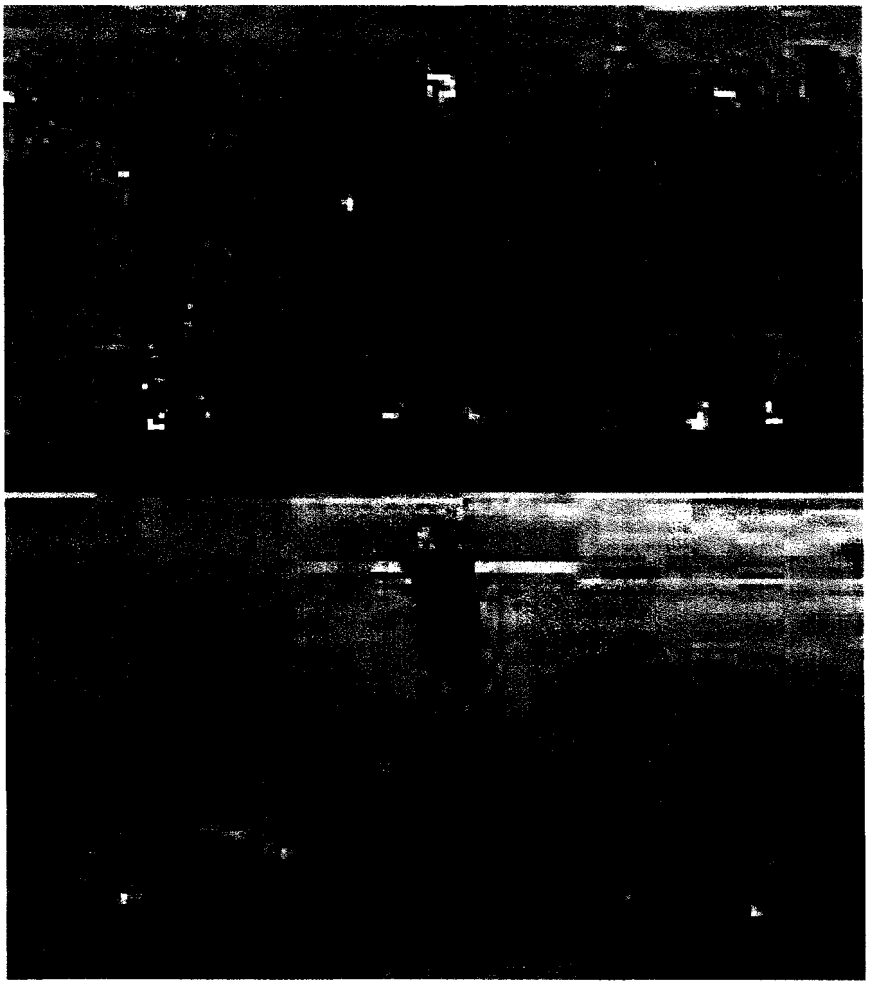


Figure 6.1: KTH data set sample frames for boxing, hand clapping, hand waving, jogging , running, walking.

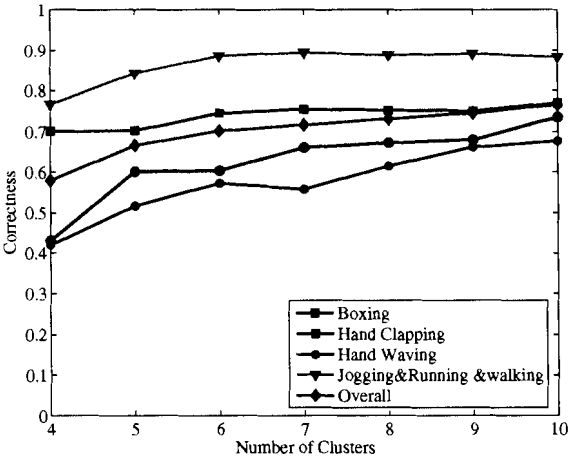


Figure 6.2: Impact of number of clusters on the performance of human action clustering for KTH data set.

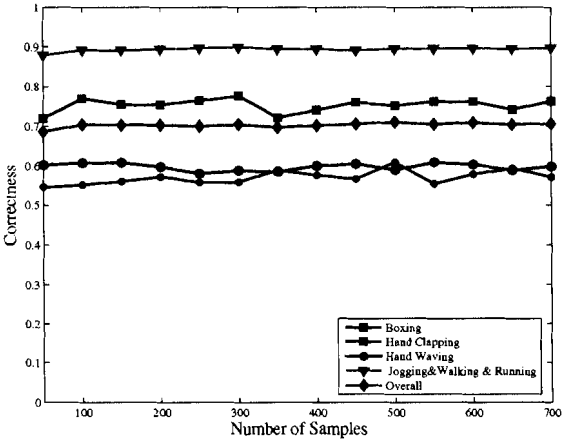


Figure 6.3: Impact of number of samples on the performance of human action clustering for KTH data set.

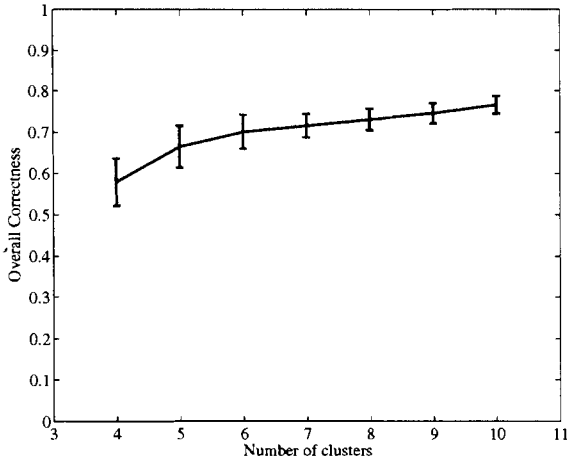


Figure 6.4: Standard deviation of overall performance for 100 runs of the code versus different number of clusters for KTH data set.

Correctness of clustering is measured by the purity of each cluster. To compute the correctness, in each round we find the number of frames for each action that has been fallen to each cluster. Then clusters are labeled with the action that has maximum number of frames in them. We do it for each cluster then add the number of frames of an action in clusters that are labeled with the same label and divide this sum by the total number of frames of each action 6.1. This number will be the fraction of actions that are correctly classified.

$$correctness(k) = \frac{\sum_{c_i=k} gt(k)}{GT(k)} \tag{6.1}$$

In this equation k shows the index for each class of action and $gt(k)$ is the number of ground truth frames which are fallen into cluster c_i and they are the majority of cluster c_i. $GT(k)$ is the total number of ground truth frames for each action in this experiments we had the ground truth for all 7640 frames of KTH data set.

Since optical flow features are very similar for walking, running, and jogging, our technique put them in the same cluster but it does not confuse these actions with boxing, hand clapping, and hand waving because the flow for them is different. In this experiment we used 100 samples from 7640 for the Nystrom extension. We ran the code 100 times and compute the average of correctness.

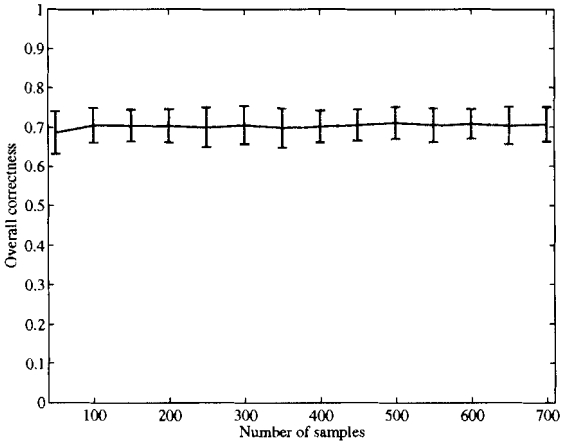


Figure 6.5: Standard deviation of overall performance for 100 rounds of the code versus different number of samples for KTH data set.

Table 6.1: Confusion table for approach [24]. There is confusion between jogging and running also hand clapping and hand waving with boxing.

	Walk	Jog	Run	Box	Hand Clap	Hand Wave
Walk	83.3	16.2	0.0	0.0	0.0	0.0
Jog	22.2	60.4	16.7	0.0	0.0	0.0
Run	6.3	38.9	54.9	0.0	0.0	0.0
Box	0.7	0.0	0.0	97.09	0.7	0.7
Hand Clap	1.4	0.0	0.0	35.4	59.7	3.5
Hand Wave	0.7	0.0	0.0	20.8	4.9	73.6

In Figure 6.4, the standard deviation of overall performance is plotted versus different number of clusters. As it is illustrated in this graph the standard deviation running the code 100 times is less than 0.06. The bars at each cluster number is the overall average correctness plus and minus the standard deviation.

We also performed another experiments to consider effect of number of samples on the correctness. The result is shown in Figure 6.3. Although there is slight changes in the performance using different number of samples for different clusters the overall performance is smooth. Again we ran the code 100 times and compute the average correctness. For this experiment we fixed the number of clusters to 6. The standard deviation for overall performance is shown in Figure 6.5 which is small.

Table 6.2: Confusion table for approach [7]. The most of the confusion is between jogging and walking or running, and between boxing and hand clapping

	Walk	Jog	Run	Box	Hand Clap	Hand Wave
Walk	90.0	4.0	5.0	0.0	0.0	0.0
Jog	20.2	57.0	24.0	0.0	0.0	0.0
Run	3.0	13.0	85.0	0.0	0.0	0.0
Box	0.0	0.0	0.0	93.0	6.0	0.7
Hand Clap	0.0	0.0	0.0	22.0	77.0	0.0
Hand Wave	0.0	0.0	0.0	7.0	4.0	85.0

Table 6.3: Confusion table for our clustering technique.

	Walk, Jog, Run	Box	Hand Clap	Hand Wave
Walk, Jog, Run	75.0	4.0	17.2	3.0
Box	3.0	66.14	2.0	28.1
Hand Clap	7.4	20.8	55.3	16.2
Hand Wave	5.8	32.3	11.6	50.2

Optical flow features are not view invariant which means that when a person walks toward different directions, the optical flow features have different values in different directions. Therefore, the similarity between walking right and left is lower than the similarity between walking in the same directions, and as it can be seen in Figure 6.2 when we are using 4 clusters the correctness for walking, jogging and running is 0.75 but when more than 5 clusters is used it is 0.85 and starts monotonically increasing with using more number of clusters. That is because we are having walking, jogging and running toward left and right in our data set which have different flows. We experimented between walking, jogging and running toward right and left using 5 clusters. We observed that always two clusters are labeled with this group of actions and each of them include up to 60 percent of these actions but in the left or right direction.

Our method is completely unsupervised compared to [7, 24] which are supervised techniques. In both of these approaches actions are analyzed by computing spatio-temporal patch features and using them for training and testing. We do not do any manual labellings to cluster the data. In Table 6.2 the confusion table from [7] and in Table 6.1 confusion table for [24] are shown. In both of these tables the column is the ground truth for the specified actions and each element a_{ij} of these matrix shows the percentage of action i that is classified as action j. As it can be seen in both of these tables, there is a confusion

between walking, jogging, and running since their features are similar. In 6.3 the confusion table for our technique using 4 clusters is shown. We run the code 100 times and for each round we computed a confusion table then we took the average which is the table 6.3.

Although they have better performance over our methods in clustering walking and running and jogging but our method is completely unsupervised and handles large data and comparing an unsupervised method and a supervised technique is not a fair comparison. To the best of our knowledge other than our method, there is no unsupervised technique using KTH dataset.

6.2 Grasshopper

Since we have to manually label the frames for evaluating the method we tested our algorithm on 3530 frames of a video of one grasshopper for the graphs in this section but we also did experiment on 80000 frames of the grasshopper video and used the clustering results for the visualization framework. The videos show that the clustering put different actions in separate clusters. Figure 1.1 shows how we set up the cameras for our experiments. We use two cameras and calibrate them by the calibration toolbox in Matlab [6]. Then, we apply the background subtraction technique to get the 2D track in each camera separately and get the 3D coordinate using the triangulation procedure in this toolbox. We smooth this track by a Gaussian filter and divide the track into non overlapping windows of size 5, $W = 5$. For each frame in this window, we compute the difference between 3D position of x and that is the feature vector or nodes of the graph. We also set δ_t in section 4.1.2 to 5 for our experiments.

We manually supply ground truth labeling of these frames into 3 classes of distinct actions – standing still, walking and jumping. We ran our code 200 times for each value of number of clusters and report the average value in the plots. The reason for this is the randomness in sampling of Nystrom method and initialization of K-means algorithm. The number of samples is 100. We compute the correctness similar to the way we did for humans and each cluster is a cluster of the action that has maximum number of frames in it. The number of ground truth frames for standing still is 1030, for walking is 2440 and for jumping is 60 which is almost all the jump frames.

The average of the correctness is shown for each of them in Figure 6.6 with respect to number of clusters. The plot shows this fraction for each action and for all of them together.

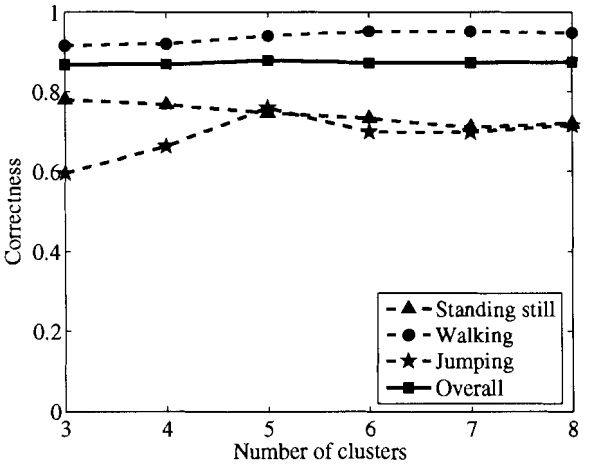


Figure 6.6: Impact of number of clusters on performance of our algorithm.

As it is shown in this figure our overall performance is above 80 percent and the graph is almost smooth for $k > 5$.

Figure 6.7 shows the importance of having samples from rare activities. In our experiments jump frames are rare and their features are very different from walking and standing still. We tested in our experiments whether the jumps are sampled or not and plotted the correctness of clustered jump frames in both cases. As it is shown in Figure 6.7, there is a big change if we do not sample jump frames. In this case, the computed eigenvectors which are the embedding coordinate will not lead to a good clustering because we estimate the eigenvectors of the whole affinity using them and if there were no samples of the unusual actions the Nystrom extension will not accurately reconstruct the eigenvectors.

We also analyze the effect of r which is the number of added jump samples on the performance of the algorithm. For each value of r, we plot the correctness of the output. As it can be seen in Figure 6.8, having more samples could result in a slightly better performance but the method is relatively stable for different values. More importantly, if we do not have any samples from the rare actions we cannot cluster them correctly.

Finally, we present the eigenvalues for 3D dominant eigenvectors in Figure 6.9. In this figure, components of the first three dominant eigenvectors are shown for each frame of the ground truth. As you can see these components get different signs for different classes of actions. For example if we consider for each frame a 3D vector that includes component of

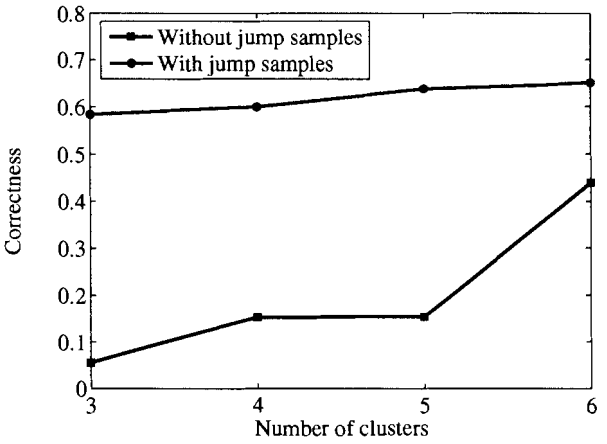


Figure 6.7: Impact of sampling from jumps on the performance. Curves show correctness of frames labeled as jumping with and without samples from this class.

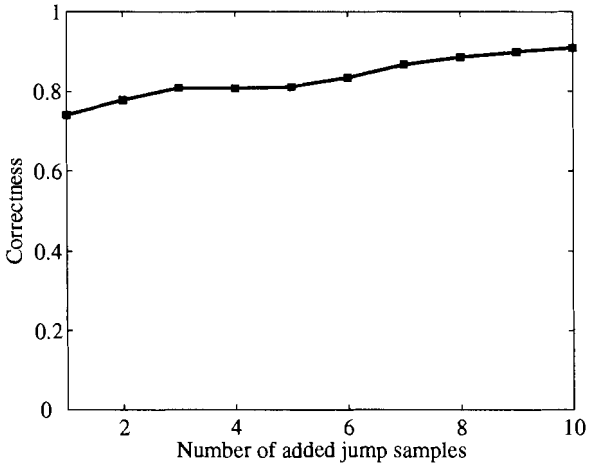


Figure 6.8: Effect of number of added jump samples on performance of detecting jump actions.

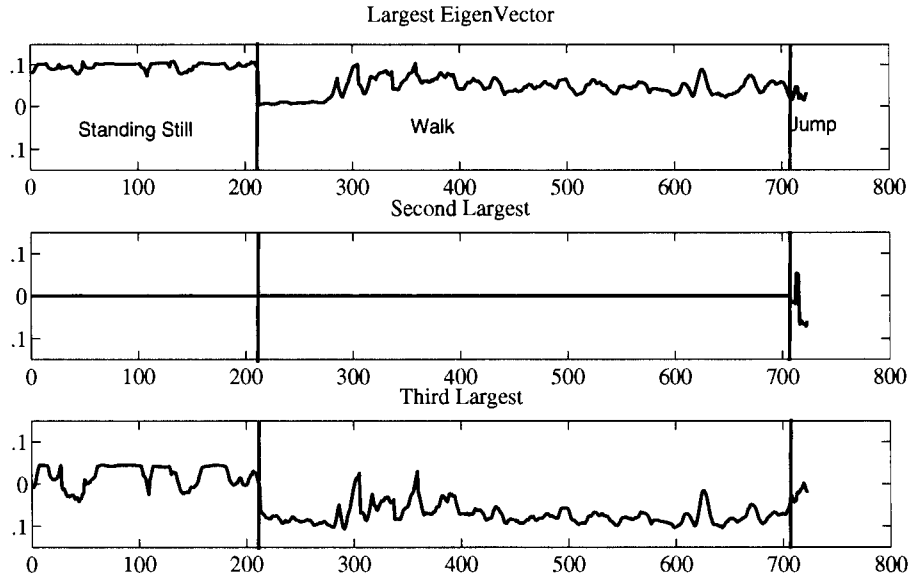


Figure 6.9: Dominant eigenvectors component values versus features for each class of action.

these eigenvector they will be different for each class of actions. For example the components of the first eigenvector are almost positive for different activities. The second eigenvector get non-zero values only for the jumps and zero otherwise. The third eigenvector is negative for nodes that correspond to walking and positive for standing still. However, this does not always hold due to the similarity between walking and standing still sequences.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis we have developed an algorithm for clustering actions of humans and animals. We experimented with our method in one data set from each domain. For humans we compute optical flow or image motion vectors as features for each figure centric image. We compute the similarity between features using normalized correlation. For the grasshopper we have developed novel features for each video segment based on the movement of the object in 3D and compute the similarity using exponential of negative of Euclidean distance. The similarity between features either for frames or video segments is the input similarity matrix for the spectral clustering.

We employed spectral clustering to cluster frames or video segments using the corresponding features. We also embedd the Nystrom extension to the spectral clustering which highly improves the performance. This is done by sampling from the original data and extending the eigenvectors of the affinity between samples to the eigenvectors of the whole affinity matrix. Using the Nystrom extension for spectral clustering for action grouping is new and it is useful because usually input videos in this field are hours of video which means many number of frames. Our experimental results showed that each of the result clusters correspond to a separate class of actions or similar actions.

Although randomness in Nystrom sampling and K-means initialization might change the results running the code different times and the correctness changes for different actions. This is the reason that we run the experiments many times. Also for humans if unusual activities present and there is no sample from them in Nystrom sampling they might not be

clustered very well. Hence, we have to add some samples from them to get good clustering for them.

For the clustering we choose the number of clusters manually and sometimes we might have less number of clusters than actions and sometimes it causes over fitting if the number of clusters is larger than action classes. Therefore, it is better to employ techniques that choose the number of clusters automatically depending on the data distribution.

Our technique is completely unsupervised which means that no manual labellings is necessary like supervised techniques during performing grouping. Although in some cases it might be desirable to have clusters labeled. There are other unsupervised techniques but our method handles large data as well as being unsupervised.

We employed an existing visualization technique for visualizing the result of each cluster in a very shorter version of the original video. We visualize the result of each cluster in a separate video which is much shorter than the input video and includes all the frames in the original one. Again since the action recognition videos are large if we use this technique to visualize all the actions and frames in a very short version it helps us to observe all the activities in a long video in a very short version of it.

7.2 Future Work

In different phases of our algorithm we could have improvements and have a better performance specially for humans action clustering.

- *Features*: In the features section for humans we used optical flow but spectral clustering with the Nystrom extension could be experimented on any other developed features for humans.

- *Similarity matrix*: For humans data set we compute a frame to frame similarity matrix and convolve it with a temporal window. Instead of that we could compute the frame to frame matrix and sum all of its elements between each sequences and obtain a scalar value that shows the total similarity between two sequences. Therefore, if the size of the frame to frame similarity matrix is n by n where n is the number of frames we could have a much smaller similarity which is $\frac{n}{k}$ by $\frac{n}{k}$ where k is the size of the sequences and it is equal to the sum of all the k by k matrices between two sequences. This will highly reduce the size of the similarity and reduce the cost of computing

eigenvectors. Then we could employ the Nystrom extension on top of that and use this for even larger data sets and this enables us to run experiments for many times in shorter amount of time.

- *Visualization*: We could also use the same visualization technique and visualize all the clusters actions in one video but giving some priority to some actions and do not visualize repetitive activities and make the video even shorter. We could also detect unusual activities and only visualize them which is useful in monitoring human activities.

We would like to experiment with our method on very long video of human actions using a stationary camera and employ the results of that for the visualization frame work.

Bibliography

[1] Tucker Balch, Zia Khan, and Manuela Veloso. Automatically tracking and analyzing the behavior of live insect colonies. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 521–528, Montreal, Canada, 2001.

[2] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *Int. J. Comput. Vision*, 12(1):43–77, 1994.

[3] Serge Belongie, Kristin Branson, Piotr Dollar, and Vincent Rabaud. Monitoring animal behavior in the smart vivarium. In *MB*, 2005.

[4] Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. In *Proc. of the Tenth IEEE International Conference on Computer Vision (ICCV'05)*, pages 1395–1402, October 2005.

[5] A. Bobick and J. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2005.

[6] Camera Calibration Toolbox for Matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[7] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *ICCCN '05: Proceedings of the 14th International Conference on Computer Communications and Networks*, pages 65–72, 2005.

[8] Alexei A. Efros, Alexander C. Berg, Greg Mori, and Jitendra Malik. Recognizing action at a distance. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 726, 2003.

[9] David A. Forsyth, Okan Arikan, Leslie Ikemoto, James O'Brien, and Deva Ramanan. Computational studies of human motion: part 1, tracking and motion synthesis. *Found. Trends. Comput. Graph. Vis.*, 1(2):77–254, 2006.

[10] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystrom method. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):214–225, 2004.

[11] D. M. Gavrila. The visual analysis of human movement: a survey. *Comput. Vis. Image Underst.*, 73(1):82–98, 1999.

[12] Zia Khan. Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(11):1805–1918, 2005. Member-Tucker Balch and Member-Frank Dellaert.

[13] Zia Khan, Tucker R. Balch, and Frank Dellaert. A rao-blackwellized particle filter for eigentracking. In *CVPR (2)*, pages 980–986, 2004.

[14] Vincent Lepetit and Pascal Fua. Monocular model-based 3d tracking of rigid objects. *Found. Trends. Comput. Graph. Vis.*, 1(1):1–89, 2006.

[15] B. D. Lucas and T. Kanade. An interative image registration technique with an application to stereo vision. In *Proc. of International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 674–679, 1981.

[16] Bruce D. Lucas. *Generalized Image Matching by the Method of Differences.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 1984.

[17] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Comput. Vis. Image Underst.*, 81(3):231–268, 2001.

[18] G. Mori, S. Belongie, and J. Malik. Efficient shape matching using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1832–1837, 2005.

[19] Maryam Moslemi Naeini, Greg Dutton, Kristina Rothley, , and Greg Mori. Action recognition of insects using spectral clustering. In *IAPR Conference on Machine Vision Applications*, pages 1–4, Tokyo, Japan, May 2007.

[20] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proc. of Neural Information Processing Systems*, 2001.

[21] E. J. Nystrom. Uber die praktische auflosung von linearen integralgleichungen mit anwendungen auf randwertaufgaben der potentialtheorie. *Commentationes Physico-Mathematica*, pages 1–52, 1928.

[22] Alex Rav-Acha, Yael Pritch, and Shmuel Peleg. Making a long video short: Dynamic video synopsis. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 435–441, New York, NY, 2006.

[23] Payam Sabzmeydani and Greg Mori. Detecting pedestrians by learning shapelet features. In *CVPR '07: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[24] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3*, pages 32–36, Washington, DC, USA, 2004. IEEE Computer Society.

[25] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[26] C. Stauffer and W.Grimson. Adaptive background mixture models for real-time tracking. In *Proc. of the 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999.

[27] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. In *Proc. of ICCV*, 1999.

[28] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[29] V. Vapnik. The nature of statistical learning theory. *Springer-Verlag*, 1995.

[30] Deepak Verma and Marina Meila. A comparison of spectral clustering algorithms. Uw-cse-03-05-01, University of Washington, 2003.

[31] Yang Wang, Hao Jiang, Mark S. Drew, Ze-Nian Li, and Greg Mori. Unsupervised discovery of action classes. In *Proc. of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[32] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

[33] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Proceedings of NIPS 2004*, 2004.

[34] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, 2000.

[35] H. Zhong, J. Shi, and M. Visontai. Detecting unusual activity in video. In *CVPR '04: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[36] G. Zhu, C. Xu, W. Gao, and Q. Huang. Action recognition in broadcast tennis video using optical flow and support vector machine. In *Computer Vision in Human-Computer Interaction*, pages 89–98, 2006.