ROBUST TRACKING OF HUMAN MOTION

DAN CALIN BUZAN

Masters Project Final Report
Computer Science Department
Graduate School of Arts and Sciences
Boston University

Submitted: May 6, 2003

Project advisor: Stan Sclaroff

ABSTRACT

This master project presents a combined solution for two problems, one: tracking objects in 3D space and estimating their trajectories and second: computing the similarity between previously estimated trajectories and clustering them using the similarities that we just computed. For the first part, trajectories are estimated using an EKF formulation that will provide the 3D trajectory up to a constant. To improve accuracy, when occlusions appear, multiple hypotheses are followed. For the second problem we compute the distances between trajectories using a similarity based on LCSS formulation. Similarities are computed between projections of trajectories on coordinate axes. Finally we group trajectories together based on previously computed distances, using a clustering algorithm. To check the validity of our approach, several experiments using real data were performed.

Contents

1 Introduction **1**

1.1 Problem Definition..1

2 Related Work **3**

2.1 Tracking Objects...3

2.2 Computing Trajectories Similarities...4

3 Estimation of Trajectories **7**

3.1 Overview and Description of Approach 7

3.2 Background Estimation...9

3.3 Blob Detection 10

3.4 Trajectory Computation 10

3.5 Camera Model...12

3.6 Tracked Features..12

3.7 3D Point Representation 12

3.8 Extended Kalman Filter Representation 13

3.9 Removing the Projection Distortion 16

4 Computing Similarities and Clustering Trajectories **18**

4.1 Overview and Description of Approach 18

4.2 Distance Metrics for Trajectories...18

4.3 Similarity Measures 19

4.4 Efficient Algorithms for Computing Similarity....................................21

4.5 Clustering...24

5 Experiments 26

5.1 Estimation and Prediction of Trajectories...26

5.2 Computing Similarities and Clustering Trajectories...............................32

6 Conclusions 44

6.1 Estimation and Prediction of Trajectories...44

6.2 Computing Similarities and Clustering Trajectories...............................44

7 Appendix 46

8 Bibliography 48

Chapter 1

Introduction

In this master project report we will present a combined approach for a computer vision problem, tracking multiple objects in 3D space and estimating their trajectories, and a mathematical–computer science problem, having a set of time sequences, define a similarity between them and cluster them according to that similarity.

The first problem, tracking multiple objects and estimating their trajectories, is still an active area of research in computer vision. Applications of the tracking methods are multiple in multiple areas: medical, biological, geographical, military, day-by-day life. For solving this problem, there are several general approaches. One of them, used in this master project report, consists of building a model of the background and detecting moving objects by frame subtraction. The same idea was used also in [1, 3, 6, 8, 9]. Another approach is to detect moving objects using image intensities (optical flow [33]). A different method is represented by the Condensation algorithm [34], which uses dynamical models together with visual observations to track moving objects.

The second problem, defining the similarity between two sequences, is a long researched topic as well. Due to its complexity there are several solutions and in general they are limited to a particular case. In computer science, for computing the similarity of sequences there are several general approaches. Euclidean distance [22,24], Dynamic Time Warping [26,27, 28, 29], Longest Common Subsequence [17, 25, 30] are examples of these general methods. Computing similarity of sequences and clustering them accordingly has a lot of applications in pattern recognition, speech recognition, ASL, stock market research, biology, music, forensics.

1.1 Problem definition

1.1.1 Estimation and Prediction of Trajectories

The problem to be solved can be defined as follows: given a video sequence (live stream or recorded) our goal is to detect if there are entities in that sequence, that are changing their spatial position. When these entities are detected we want to be able to estimate their spatial position and velocity (within certain limits) such that if certain events occur (an entity is temporarily obscured or disappears shortly from the sequence) we still want to be able to predict its trajectory and behavior until the event will eventually stop.

We will not make any assumption regarding the type of the scene recorded in the video sequence. It can be a simple room, a square, a street corner or just synthetic data. We will need also to find a model, that is able to emulate the motion of an object in 3D space, which is robust enough to withstand to various influences exerted by the

environment. The last requirement is dictate by the fact that motion of an object in 3D space is influenced by various factors, so our object in 3D space will need to train the model with observed motion parameters. In addition to this, our model will have to take into account the noise injected by the capturing device.

At the end of the process, our model should provide us with an estimation of the trajectory of the object whose motion it tried to emulate.

1.1.2 Computing Similarities and Clustering Trajectories

The problem in this case consists in finding a meaningful similarity between two time sequences (trajectories). Moreover, this similarity should allow us to group trajectories together and, eventually, to index them.

In our system we look for a similarity that has certain flexibility. This similarity should not depend on the spatial distance between trajectories. Nonetheless, it should depend on the rotation vector. Another feature that we want in our similarity definition is to handle outliers efficiently. This requirement is motivated by the first problem. Because of the noise generated by the capturing device some sequences may contain errors. If the amount of errors in a trajectory is relatively small, it is important to be able to decrease its influence during similarity computation. Moreover, because we are dealing with entities that are moving with different speeds, our similarity definition of trajectories should take this issue into account too.

Capturing device main have a variable sampling rate or simply, may fail for some periods of time during data collection. This issue should be covered by our similarity definition as well.

Finally, the clustering algorithm that we use to group the trajectories together should satisfy some requirements: the clusters that it generates should be sound relative to the similarity definition.

Chapter 2

Related Work

2.1 Tracking Objects

A method for tracking people and interpreting their behavior is described by Wren et al. [9]. The background is modeled using a Gaussian distribution and detected blobs are tracked using a Kalman filter. Once a blob is detected, the system analyzes it and identifies the head, hands and feet locations.

Stauffer et al. [1] build a background model using a mixture of Gaussians. The mixture of Gaussians gives an increased flexibility to their model because it allows sudden changes in the background (shadows, a flag in the wind, a construction flasher). To establish the correspondences between blobs in consecutive frames the authors employed a linearly predictive multiple hypothesis algorithm. The moving objects (cars, people) in the video sequence are tracked using Kalman filters.

Siebel et al. [5] built a system for tracking people. They use an active shape model for the contour of a person in a video frame. Tracking is done with the help of Kalman filtering using second order motion models for human motion. The state of the tracker includes the current outline shape (as a point in the PCA space S of trained pedestrian outlines), which is updated as the observed outline changes during tracking. In order to improve detection the authors implemented a color filter using the luminance (Y) subspace to compute the differences between current frame and background.

Haritaoglu et al. [7] created a real time visual surveillance system for detecting and tracking people and monitoring their activities in an outdoor environment. Background is modeled using three values for each pixel: minimum and maximum intensity and the maximum intensity difference between two consecutive frames. To track people the system employs a second order motion model for each object. The matching strategy consists of two stages: estimation of object displacement and binary edge correlation between the current and previous silhouette edge profiles.

Sato et al. [2] present a methodology for tracking moving persons and vehicles in outdoor image sequences. They use slant cylinders for modeling the detected moving objects (people and vehicles). After detecting the moving objects in a frame they apply to them a temporal spatio-velocity transform (a windowing operation over the binary image sequence followed by a Hough Transform). The result of this transformation is a four-dimensional image sequence (TSV image). The system tracks the objects by fitting the

elements of the TSV image into spatio-temporal slant cylinder models that represent moving objects in the spatio-temporal image cube.

Rosales et al. [8] describe an approach for tracking moving people and recognizing their actions. The background is modeled using a Gaussian distribution. Detected blobs are tracked using an extended Kalman filter. The obtained 3D trajectories along with other information (occlusion and segmentation information) are used to extract stabilized views of the moving objects. Those views are subsequently used as input to action recognition modules.

The approach used by Ellis et al. [6] for motion detection and tracking of pedestrians in an outdoor environment consists of building a background model using a mixture of Gaussians. To minimize the effects of sudden changes in scene illumination they compute two foregrounds (using an intensity-based model and a color-based model) and combine the results thereafter. Their system is based on a priori learning of the scene, and all the scene features are stored in a database. Detected objects are tracked using a Bayesian network.

A method for tracking moving objects in an outdoor environment is presented also by Zhou et al. [3]. Their system classifies blobs into three categories: single person, group or vehicle. Background subtraction is performed using color and texture (edge difference using the Sobel Gradient operator). For each blob, several features are computed (centroid, length, width, area, compactness, orientation, motion direction). Objects are tracked using a simple matching procedure based on centroid, shape and color matching.

2.2 Computing Trajectories Similarities

Different types of sequences (temporal, spatial, spatio-temporal sequences, and various kind of biological or chemical structures) are encountered in a wide range of domains (pattern recognition, genetic engineering, chemistry, etc). When the number of sequences that is available in a particular application becomes large, it is important to have efficient and reliable methods for indexing such sequences and querying them. An essential part of designing the querying process is computing the degree of similarity of two sequences.

Examples of these kinds of sequences are:

- pattern of growth of a company
- selling pattern of a product
- price variation of a stock
- spectrogram of a person's speech
- musical score of a song
- bird migration pattern

- DNA sequence of an individual
- fingerprints of an individual

Regarding temporal data sequences, there have been several efforts to design a model of this kind of sequences, to design languages to query such data and to create data structures that allow efficient processing of these queries. There are several measures defined and several techniques used for computing similarity of two trajectories.

Agrawal et al. in [24] proposed a method for indexing time sequences and processing similarity queries of these sequences. This method used Discrete Fourier Transform (DFT) to map time sequences from spatial domain to frequency domain. The bases of this approach are that, for most sequences of practical interest, only a few frequencies are strong, and that the Fourier transform preserves the Euclidean distance in the frequency domain. The similarity measure chosen is the Euclidean distance because of its usefulness in several cases and because it can be used in a flexible manner: many different measures can be expressed as the Euclidean distance between feature vectors in some feature space, and different choices of feature spaces lead to different similarity measures.

Faloutsos et al. in [22] devised an algorithm, *FastMap*, whose main idea is to reduce dimensionality of objects involved either in indexing or in querying. Given N points in an n-dimensional space, the algorithm will try to project them on k mutually orthogonal directions such that the relative distances between original objects will remain the same for their projections in the k-d space. The similarity measure used in the experiments (for documents similarity) is cosine similarity of the projection vectors associated with the documents (cosine of the angle between the two vectors).

Agrawal et al. in [19] presented a model to compute the similarity of time sequences. This model is based on the idea that two time sequences are similar if they have enough non-overlapping, time-ordered pairs of subsequences that are similar. In the process of computing the similarity, one of the two sequences can be scaled by any suitable amount and translated adequately before establishing its subsequences that match the subsequences in the other sequence. Two subsequences are similar if one lies within an envelope of width ε drawn around the other one, ignoring outliers.

Yi et al. in [28] presented an improved version of *FastMap* algorithm of [22]. Their model combines *FastMap* with Dynamic Time Warping (DTW). The similarity of time sequences is computed using DTW (while *FastMap* is used to select time sequences that are close enough to the query argument). Kim et al. in [27] also use DTW for computing the similarity of time sequences. The difference from the paper of Yi consists in how the time sequences are indexed in the database and how the potential candidates are selected during a query computation.

Keogh in [26] described also a procedure for indexing time sequences and computing similarity between time sequences using Dynamic Time Warping. The approach is similar to those presented in [28] and [27]: finding a lower bound function that will allow finding quickly potential candidates for a range query. Keogh defined a lower bound function based on piecewise constant approximation [29].

Yazdani et al. in [25] proposed a method for indexing and computing the similarity of sequences using the Longest Common Subsequence (LCSS) (described in [30]) as a similarity measure. Sequences are indexed using the first r moments around the mean (these moments will be the components of the feature vector associated with a sequence). During the computation of a range query, the candidates are selected using the LCSS method.

Das et al. in [17] developed a model for computing the similarity of time series. This model is using LCSS for computing the similarity measure, which is expressed as a triple (F, γ, ε) where F is a set of transformation functions that map integers to integers, γ is a parameter that controls the ratio of sequences length and ε controls the size of the interval where the mapped value should be.

Chapter 3

Estimation and Prediction of Trajectories

3.1 Overview and Description of the Approach

This chapter describes the modus operandi of our system: from processing the frames to trajectory computation. The approach presented here is based on large parts on the method previously described by Rosales et al. in [8]. The differences between these two approaches will be illustrated further while presenting the overview of our technique.

A graphical presentation of our model is exhibited in Fig. 3.1. First step consists of initializing the system by collecting enough video frames for computing statistical measurements associated with our background model. These measurements are the mean and covariance of each image pixel in 3D color space. The reason behind collecting these measurements is the need to have a representation of the background elements, representation that will help later to extract the foreground. A new element in our approach, comparative to [8], is that if we have enough information about scene topology (coordinates of at least four points located in the ground plane) we can compute a homography that will help us to remove the projective distortion of computed trajectories.

After finishing the initialization part, the moving objects in the scene are segmented using a maximum likelihood estimation. Next, a connected component algorithm is employed to convert the foreground pixels that we got after segmentation into entities, blobs. To eliminate the noise and artifacts generated by the segmentation process we apply morphological operations followed by a size filter.

The blobs that we get at this point represent the moving objects in the scene. Nevertheless, the results that we obtained may contain inaccurate information because of occlusions or increased similarities between the background and objects. In order to gain reliable estimates of the blob motion trajectories, we employ an Extended Kalman Filter (EKF) to predict the position of a blob when occlusions (with other blobs or elements of background) occur. Finally, the trajectories associated with the moving objects are built step by step by matching blobs in the previous frame with the blobs in the current frame. These trajectories will represent the output of this part of our system. Another difference with the approach presented in [8], is the matching process between blobs in two consecutive frames. Our method allows, in cases of uncertainty, to follow two hypotheses in parallel until a decision can be made.

The EKF formulation is similar to the one presented in [8] and [31] and is based on first order Newtonian dynamics. This formulation embeds the focal length of the camera with the depth similar to the method presented in [10]. Objects are considered without depth and the features that are tracked are the opposite corners of the object's

bounding box. Our EKF formulation provides an estimate for position and velocity in 3D of each object's bounding box. The input to EKF is the bounding box that includes the moving object. Noisy measurements, occlusions, acceleration of objects are events that EKF is able to handle, due to information stored.

The last step, which is applied only if we have information about scene topology, consists of removing the projection distortion of a trajectory by applying the homography computed in the initialization step of our system.

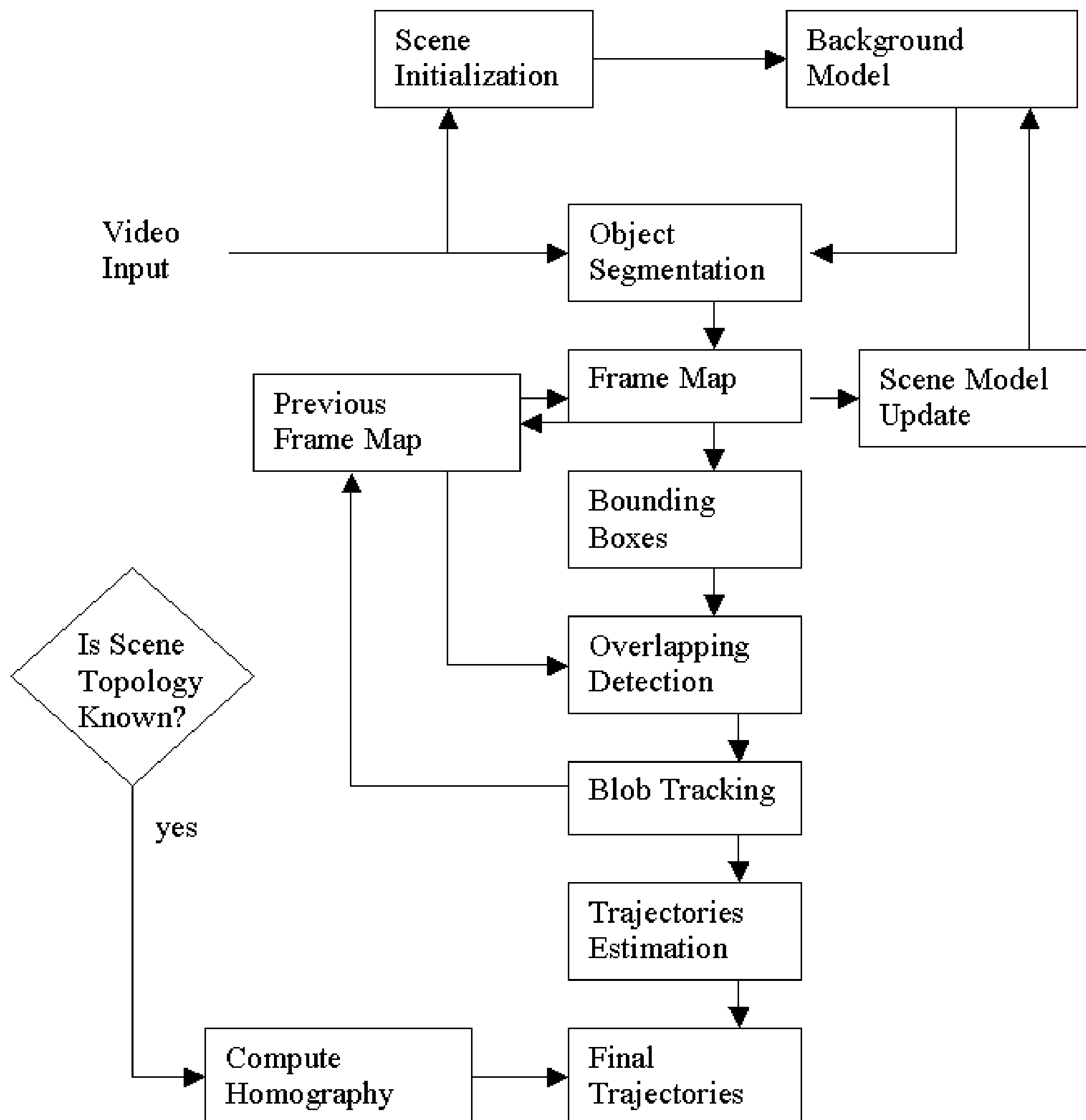


Fig 3.1 Process Diagram

3.2 Background Estimation

An essential step in computing correct trajectories is the process of building a model for the background. There are several ways for building a background model, described in the previous chapter. In our system we model the background as a Gaussian distribution. The reason for choosing this approach is that it offers a robust and reliable model despite the noise induced by the capturing device or occasional moving objects.

For computing the statistics of background pixels we use 100-200 frames (3-6 seconds of video). The number of frames used in computation depends on the scene type: for indoor scenes we use 3 seconds of static background. For outdoor scenes, it is more difficult to get static background. To compensate for this, we use a bigger number of frames for outdoor scenes, so that even if there are moving objects in the scene, the increased number of samples for each pixel will minimize the "noise" effect introduced by the passing objects. For this case (of outdoor scenes), in addition to the background model we also compute a supplemental structure that stores several features for each pixel: histogram, min value, max value, index of the histogram element with the maximum value, and indexes of histogram elements that are enclosing a certain percentage of the total number of pixels (75% and 95%).

In both indoor and outdoor cases, for each pixel p we compute its mean $\mu_p = (r_p, g_p, b_p)^T$ in color space. The covariance of the color distribution for the pixel p is then estimated using N frames:

$$K_p = \frac{1}{N-1}\sum_{t=1}^{N}(I(p,t)-\mu_p)\cdot(I(p,t)-\mu_p)^T, \tag{3.1}$$

where $I(p,t)$ represents the intensity of pixel p in the tth frame.

The background model (K, μ) is updated periodically to account for scene changes: shadows, lights, moving objects that stop moving in the scene and stopped objects that start moving. In this implementation only the mean is updated using an adaptive filter.

$$\mu(t) = [(1-\alpha)\cdot I(t-1) + \alpha\cdot\mu(t-1)]\cdot M_{t-1} + [(1-\eta)\cdot I(t-1) + \eta\cdot\mu(t-1)]\cdot \overline{M}_{t-1}, \tag{3.2}$$

where M_t is the binary map associated with the tth frame, $\overline{M}_t$ is the complement of the M_t binary map and α is a learning rate indicating how much of the current background occupied by background pixels in the current frame should be preserved and η is another learning rate indicating how much of the current background occupied by foreground pixels in the current frame should be preserved. Both α and η are close to 1. Multiplication in the equation (3.2) is done in a pixel-by-pixel form. The way that M_t is computed will be described below.

3.3 Blob Detection

When the background model is computed the system is ready to track moving objects. This is done by computing a distance between the current frame and the background using the log-likelihood measure:

$$d_p(t) = -\frac{1}{2} \bullet \left[\left(I(p,t) - \mu_p(t) \right) \cdot K_p^{-1} \cdot \left(I(p,t) - \mu_p(t) \right)^T + \ln\left| K_p \right| + n \cdot \ln(2 \cdot \pi) \right], \qquad (3.3)$$

where n represents the dimension of color space. The magnitude of the distance determines if the pixel p in the current frame belongs to the background or to the foreground. The set of all $d_p(t)$ distances generates an intermediate image. This intermediate image is a gray scale image and it is further segmented generating the difference image FD_t associated with the current frame t.

After computing the difference image, several basic operations are performed on it. First, a closing operation (dilation–erosion) to eliminate small holes and regions. Second, a connected component analysis is performed on FD_t giving us the set of blobs b_i:

$$b_i(t) = ConnCompAnalysis(\bigcup_p \left[d_p(t) < \Gamma \right], i), \qquad (3.4)$$

where Γ is a threshold (that will help us decide which pixel is background and which pixel is foreground) and function ConnCompAnalysis(FD_t, i) will return the ith blob existing in the tth frame. Finally, the last operation applied is a size-filter. This operation will remove small blobs that represent either objects that are too far or noise introduced by the caption device. This final image, which we get after applying all these operations, represents the binary map M_t associated with the tth frame.

3.4 Trajectory Computation

Computation of trajectories of moving objects is done using the binary maps M_t M_{t-1} corresponding to the current and the previous frame. Trajectories are computed by trying to match each blob $b_i(t)$ of binary map M_t with one or more blobs $b_l(t-1)$ of M_{t-1}. In order to be able to estimate and predict the blob trajectory, we assign to each blob a tracker unit T_j. The method for matching blobs is described below:

a. $b_i(t)$ is matched with $b_l(t-1)$ if their bounding boxes overlap and no other overlapping occurs.
b. if $b_i(t)$ overlaps several blobs $b_l(t-1)$ $l = l_1, l_2, l_3, \ldots$ then $b_i(t)$ is matched with all these blobs. Such an occurrence may be caused by different types of events: blobs

from the previous frame may represent different objects that are too close, or blob $b_i(t)$ represents a single object that was obscured by the background elements in the previous frame.

c. if $b_i(t)$, for $i = i_1, i_2, i_3, \ldots$ overlaps $b_l(t\text{-}1)$ then all these $b_i(t)$ blobs are matched with $b_l(t\text{-}1)$. As before, this situation may occur in different circumstances: $b_l(t\text{-}1)$ may have been a group of objects that are splitting in the current frame into individual objects, or blob $b_l(t\text{-}1)$ represents a single object which is overlapped by elements of the background in the current frame.
d. if $b_i(t)$'s bounding box doesn't overlap anything than there are also several possible explanations: the blob may represent a new object in the scene, or an existing object that was occluded by the background, or an object that had a sudden change in speed or direction; in this last case, a search is performed in the binary map $M_{t\text{-}1}$ to find a possible match.
e. if $b_l(t\text{-}1)$'s bounding box doesn't overlap anything, then as above, the possible hypotheses are: the object disappeared from the scene, the object was occluded by background or the object had a sudden change in speed or direction; in this last case, there is a search in binary map M_t for finding a possible match.

The matching process is assisted by the tracker units T_j associated with every moving object. The goal is that the tracker associated with each object will keep this association until the object disappears from the scene (despite the possible occlusions with other objects or with the background). A tracker unit is designed using the model described in [08]. Each tracker will hold information about the blob or blobs that are associated with it. This information is: corners of the bounding box, depth (up to a given constant), probability of a collision within certain time limits, hypothesis about the current blob, duration of tracking.

For every new object that appears in the camera field view, a new tracker unit T_j will be associated with it. At every frame, blobs from the current binary map M_t are matched against blobs belonging to previous binary map $M_{t\text{-}1}$. The tracker's role is important when a collision between two blobs occurs, when a blob splits or when a blob is occluded by parts of the background. In the cases described above, the information stored in a tracker unit will help us decide what hypothesis to follow.

There are some unresolved issues associated with the trackers: when a tracker unit is associated with a newly appeared object, it needs several frames until it stabilizes. Also, a special case is when an object enters the scene immediately after another object has left the scene.

3.5 Camera Model

For this project we use a 3D central projection camera model similar to that used in [8, 10]. The equation model is:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} X_C \\ Y_C \end{pmatrix} \cdot \frac{1}{1 + Z_C \cdot \beta} \tag{3.5}$$

where (X_C, Y_C, Z_C) is the 3-D location of a point in the camera reference system, (u, v) is the image location of the projection, and $\beta = 1 / f$ is the inverse focal length. Also, the origin of the coordinate system is set to the center of the image plane.

One important property of this model is that it decouples the representation of the camera from the representation of depth. Thus, altering the inverse focal length (β) alters the imaging geometry independent of the depth of the object. Also, this model is numerically defined even in the case of orthographic projection (when $\beta \to 0$).

3.6 Tracked Features

In order to reduce the complexity of computation, two feature points are tracked per blob: the opposite corners of the blob's bounding box. The reason for doing this is to avoid searching for and matching too many features in consecutive frames.

Even though the blob model is non-rigid, it is expected that the sizes of an object will not change too much from one frame to another.

3.7 3-D Point Representation

A "feature point" is defined in terms of its image location in the first frame in which it appears. Tracking this feature results in a series of measurements of image location of this feature in the subsequent frames.

For this application we use a representation of 3D points similar to the one used in [8], [10] and [11]. A point in 3D is parameterized using the x and y coordinates while the depth z will be combined with the inverse of focal length: $(x, y, z\beta)$. Similarly, the speed of the object relative to the camera is represented by the vector: $\left(\dot{x}, \dot{y}, \dot{z\beta} \right)$.

We assume that the objects are performing a linear translational motion in 3D. Using the Extended Kalman Filter described in Section 3.7, allows extracting the relative 3D position of the object using several frames. This formulation allows us to compute the depth of the object up to a scale factor. For certain cases, when we know coordinates of enough points in the image plane, we can compute a better value for the object depth.

These cases are presented in Section 3.8. Even in these cases the depth may not be accurate if the moving object is above or below the ground plane (we assume that the bottom part of the bounding box of an object is at the ground plane level).

The reasons for using this representation have been presented in [10] and [11], and we briefly summarize them here. The sensitivity of an object's image location to object motion is similar along the x-axis and y-axis in the 3D coordinates, while the sensitivity diminishes along the z-axis. Moreover, this sensitivity decreases with the increase of the focal length of the imaging geometry. In the extreme case of orthographic projection there is zero image plane sensitivity to z-axis motion.

$$\begin{pmatrix} X_C \\ Y_C \\ Z_C \cdot \beta \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \\ t_z \cdot \beta \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \beta \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \qquad 3.6$$

Equations (3.5) and (3.6) show that $t_z \cdot \beta$ is enough to compute image plane measurements as a function of object coordinates and camera parameters. Another advantage of this model is that the motion sensitivity along the z-axis does not degenerate for long focal lengths. Using the equations 3.5 and 3.6 we get

$$\frac{\partial u}{\partial t_z} = \frac{-X_C \cdot \beta}{(1 + Z_C \cdot \beta)^2} \text{ and } \frac{\partial u}{\partial (t_z \cdot \beta)} = \frac{-X_C}{(1 + Z_C \cdot \beta)^2} \qquad 3.7$$

The above equations show that motion along z-axis is visible even for long focal lengths when we combine depth with focal length, while when using depth alone that motion will decrease significantly. This become obvious when we study the behavior of Eq. 3.7 when focal distance becomes infinite and consequently $\beta \to 0$.

3.8 Extended Kalman Filter Formulation

The Kalman filter is a set of mathematical equations that provides an efficient solution to the least-squares method. Since Kalman publish his paper in 1960, there has been a lot of research in this area and a lot of applications and extensions were developed expanding Kalman's work.

For our system we use a novel extended Kalman Filter (EKF) formulation to predict recursively the future positions and velocities of moving objects. Prediction is computed using the current positions and velocities of the objects. The model that we use here is similar to the one used in [8] and described in [31].

We are using a first order EKF. The reason for this choice is that the underlying process is inherently nonlinear. In theory, motion of mechanical systems can be considered to have constant velocity or constant acceleration. Nonetheless, no real object

can start with acceleration zero and end in a constant acceleration without changing the acceleration continuously. On top of that, human motion hardly can be considered as having constant velocity or acceleration.

For a better design of our system we considered that objects are moving in 3D space (x, y, $z\beta$), hence having a 3D trajectory. This decision was motivated by the fact that due to the perspective foreshortening effects, even when the trajectories are linear, trajectory will appear as non-linear, unless the object is moving parallel to the image plane. Therefore, a 3D trajectory model offers an improved performance and an increased simplicity over models that are using a 2D trajectory model.

The tracked objects are considered to be planar objects with no depth. Also, because the sizes of tracked objects are small relative to the size of view plane we will not track the rotation motion. So, the state vector considered is:

$$X = \left(x_0, y_0, x_1, y_1, z\beta, \dot{x}_0, \dot{y}_0, \dot{x}_1, \dot{y}_1, z\dot{\beta} \right)^T \quad 3.8$$

where $(x_0, y_0, z\beta)^T$ and $(x_1, y_1, z\beta)^T$ represents the corners of the object's 3D bounding box (features that we are tracking). Vectors $\left(\dot{x}_0, \dot{y}_0, z\dot{\beta} \right)^T$ and $\left(\dot{x}_1, \dot{y}_1, z\dot{\beta} \right)^T$ represents the velocities of the bounding box corners relative to the camera. The measurement vector is defined as:

$$z = (u_0, v_0, u_1, v_1)^T \quad 3.9$$

where u_0, v_0, u_1, v_1 are the image plane coordinates of the tracked features. The measurement vector is associated with the state vector through the measurement equation: $z_k = h(x_k + v_k)$, where v_k represent the measurement noise. In our approach h(•) is a non-linear function (Eq. 3.5) and v_k is considered to be additive.

The 3D trajectory and velocity are recovered up to scale factor (in certain cases, described in Section 3.8, we are able to find a better solution for the 3D trajectory and velocity). The unique image plane trajectory of a tracked object represents the projection of a set of possible solutions of the Kalman Filter equations. Consequently, given the motion of an object into the (x, y, $z\beta$) space, we can predict its future positions in the image plane.

The recursive part of our EKF formulation is presented below. Before starting the tracking process we need to initialize the state vector $\hat{x}_0^-$ as well as P_0^-, R_0 and Q_0. If the values of $\hat{x}_0^-$ and P_0^- are incorrect this may lead to numerical problems or divergence. More details about these problems are presented in [31]. Also in [12] is described a method for initializing P_0^-, R_0 and Q_0.

Time Update ("Predict")
1. Project state ahead

$$x_{k+1} = A_k \cdot x_k + w_k \qquad 3.10$$

where x_k is the state vector at time k, w_k is the process noise (and is considered to be additive), A_k is the system evolution matrix, the matrix that describes how a new position at moment t_{k+1} depends on the previous position and velocity at the moment t_k, and how the velocity at time t_{k+1} relates to previous velocity. In our case, A_k is based on first order Newtonian dynamics in 3D and is assumed to be time invariant ($A_k = A$).
The EKF time update equation becomes:

$$\hat{x}_{k+1}^{-} = A \cdot \hat{x}_k \qquad 3.11$$

2. Project error covariance ahead

$$P_{k+1}^{-} = A \cdot P_k \cdot A^T + W \cdot Q_k \cdot W^T \qquad 3.12$$

where Q_k is the process noise covariance and W is the Jacobian matrix of the transformation A with respect to w.

Measurement Update ("Correct")
1. Compute the Kalman gain (blending factor)

$$K_k = P_k^{-} \cdot H_k^T \cdot (H_k \cdot P_k^{-} \cdot H_k^T + V_k \cdot R_k \cdot V_k^T)^{-1} \qquad 3.13$$

where R_k is the measurement noise covariance at the moment k, V_k is the Jacobian matrix of function $h(\bullet)$ with respect to v. Matrix K_k acts like a blending factor in the update estimate $\hat{x}_k^{-}$. If the measurement error is large, then the covariance matrix R_k influences K_k such that some components of K_k get small. Hence, the influence of the measurement z_k in the following step, update estimate, gets small.

2. Update estimate with measurement z_k

$$\hat{x}_k = \hat{x}_k^{-} + K_k \cdot (z_k - h(\hat{x}_k^{-}, 0)) \qquad 3.14$$

3. Update the error covariance matrix

$$P_k = \left(I - K_k \cdot H_k\right) \cdot P_k^{-} \qquad 3.15$$

Here H_k is the Jacobian matrix of $h(\bullet)$ with respect to x:

$$H_k = \begin{pmatrix} \frac{1}{\lambda} & 0 & 0 & 0 & -\frac{x_0}{\lambda^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\lambda} & 0 & 0 & -\frac{x_0}{\lambda^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\lambda} & 0 & -\frac{x_0}{\lambda^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\lambda} & -\frac{x_0}{\lambda^2} & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad 3.16$$

where $\lambda = 1 + z\cdot\beta$.

Other assumptions are: w and v are random vectors, independent of each other, white and with normal probability distributions:

$$\begin{aligned} p(w_k) &\sim N\left(0, W \cdot Q_k \cdot W^T\right) \\ p(v_k) &\sim N\left(0, V_k \cdot R_k \cdot V_k^T\right) \end{aligned} \quad 3.17$$

3.9 Removing the Projection Distortion

When the matching process described above ends, the result of it is several pairs of blobs and tracker units ($b_i(t)$, T_j). All these pairs are saved in a table for further processing. In certain cases (most of indoor scenes, some of outdoor scenes) it is possible to apply a projective transformation (a homography) that will convert the trajectory points from the view plane to the ground plane. The goal of this operation is to remove the projection distortion from the perspective image of the ground plane.

In order to be able to build this homography we need to find at least 4 points on the ground plane (a homography has 8 DOF). Using their coordinates we can compute the homography matrix H: if there are 4 points, we will have to solve a system of 8 equations with 8 unknowns, if there are more then 4 points, we will have to estimate H using the least square method (see [13]).

$$H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \quad 3.18$$

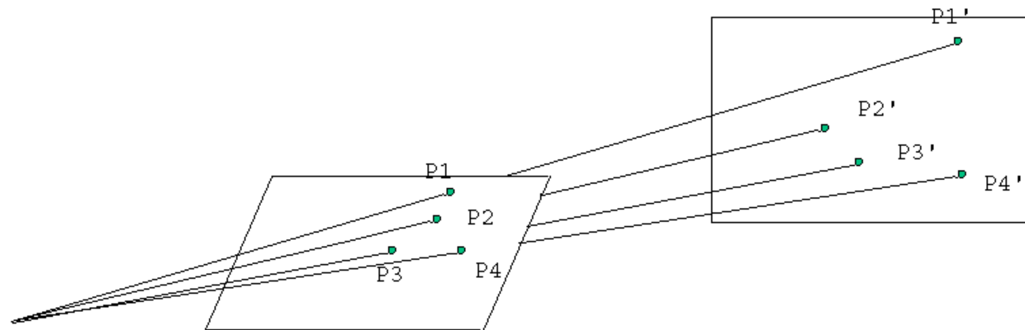


Fig 3.1 Homography

After we finish computing *H*, the transformation applied to trajectory points from the view plane is described by equation (3.7), below:

$$p_i' \equiv H \cdot p_i \qquad 3.19$$

Chapter 4

Computing Similarities and Clustering Trajectories

4.1 Overview and Description of the Approach

This chapter describes the processing of trajectories that have already been computed using the methods in the previous chapter. The final goal will be to group trajectories that have a certain degree of similarity.

The approach presented here is based on the work done by Vlachos et al. in [16]. We will define a similarity measure for the trajectories based on the Longest Common Subsequence (LCSS) algorithm [30]. Using this measure we will compute the distances between previously computed trajectories. The main difference between our approach and the method presented in [16] is that we compute the distance between two time sequences as a pair of numbers. Each number will represent the similarity between the projections of the two time sequences on coordinate axis. The effect of this change is a decreasing in computation time, from a cubic time to a quadratic one. Also, another difference appears in definition of LCSS, where we are using a threshold to limit the ratio between the lengths of the two time sequences.

The last step will consist of grouping the trajectories into clusters. In our approach we use are using a hierarchical algorithm presented in [14].

4.2 Distance Metrics for Trajectories

In general, trajectories are recorded during a tracking session with the aid of capturing sensors. In addition to noise introduced by capturing devices, there are other artifacts created by lights, weather, background, etc. All these factors are introducing outliers that will change the features of a trajectory.

There are several popular methods used for computing the distance between sequences. The Discrete Fourier Transform (DFT) is used for feature extraction because preserves the Euclidean distance between sequences. Unfortunately it works only for sequences that have the same length. The Euclidean distance and Dynamic Time Warping (DTW) are popular choices too, but they have drawbacks that are presented below:

Our goal is to define a distance between trajectories that will take into account the following factors:

1. Different sampling rates and different speeds – a sensor that collects data may fail in doing that for a certain amount of time or may vary the rates at which it is collecting data (e.g. a camera that collects data at variable frame rate). Also, two objects that are moving on the same trajectory but having different velocities will generate two time sequences that are different. In this case Euclidian distance will fail to compute an adequate result.
2. Similar motions in different space regions – two trajectories that are similar but are positioned in space in different areas (e.g. particles in a magnetic field, similar evolution of stock prices that have different values). For this case, to overcome this problem, we subtract from each trajectory the average value.
3. Outliers – these errors in data collecting are caused by several factors. Unfortunately the outliers have a big impact (even if they change the trajectory in a few points) when we compute distance between two time sequences using either Euclidean distance or DTW. One of the major disadvantages of Euclidean distance and DTW is the fact that they try to match all the points of the trajectories.
4. Different lengths – Often it happens that time sequences have different lengths (or their lengths are truncated because of external factors). In this case we cannot use Euclidean distance because it is assuming that the trajectories have equal length.
5. Efficiency – We want to devise an algorithm that will perform the similarity computation fast.

An algorithm that matches the above requirements (except the second) is presented in [30]. The algorithm, Longest Common Subsequence (LCSS), is at the basis of the algorithm used in this master project report. The reasons for choosing this algorithm are:

- it allows to match two sequences by stretching them, without rearranging the order of the elements, but allowing some elements to be unmatched
- sequences from scientific domains are composed of elements that are real numbers, obtained during collecting data, using devices with limited precision. Consequently the elements of these sequences should be matched based on proximity.
- The LCSS model allows an efficient approximate computation (presented in Section 4.3)

The algorithm presented here will extend LCSS in 3D (motion will be in a plane, while the third dimension will be time). The algorithm will also handle the second requirement (handling similar motions in different space regions).

4.3 Similarity Measures

This section will cover the similarity model. The model is based on the model presented in [16]. One of the differences is that instead of computing the distance as a single value, we will compute the distance as a vector (D_x, D_y), where D_x and D_y

represent the distances between trajectory projections on the motion axes. Motion is considered to be in 3D space. The axes that define the motion plane will be called motion axes, while the third axis will be called time axis. We represent objects that are moving in the motion plane as simple points and we assume that the time is discreet.

Let A and B two data sequences with sizes n and m respectively. $A = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n}, a_{y,n}))$ and $B = ((b_{x,1}, b_{y,1}), \ldots, (b_{x,m}, b_{y,m}))$. The projection of A on the x-axis will be denoted as $A_x = (a_{x,1}, \ldots, a_{x,n})$ and the projection of A on the y-axis will be denoted as $A_y = (a_{y,1}, \ldots, a_{y,n})$. Also, we will use the functions $Head(A)$ defined as $Head(A) = ((a_{x,1}, a_{y,1}), \ldots, (a_{x,n-1}, a_{y,n-1}))$ and $Head(A_x) = (a_{x,1}, \ldots, a_{x,n-1})$.

Definition 4.1: Given an integer δ and real numbers $0 < \varepsilon$ and $0 < \rho \leq 1$, we define $\mathrm{LCSS}_{2D}(\delta, \varepsilon, \rho, A, B)$ as follows:

$$\begin{gathered}\mathrm{LCSS}_{2D}(\delta, \varepsilon, \rho, A, B) = (LCSS_{\delta,\varepsilon}(A_x, B_x), LCSS_{\delta,\varepsilon}(A_y, B_y)), \text{ or} \\ \mathrm{LCSS}_{2D}(\delta, \varepsilon, \rho, A, B) = 0 \text{ if } \min\{m, n\} < \rho * \max\{m, n\}\end{gathered} \quad 4.1$$

where ρ is a constant named Length Aspect Ratio (LAR) that controls the difference in size between the trajectories and $LCSS_{\delta,\varepsilon}(A_x, B_x)$ is defined as follows:

$$\begin{cases} 0 & \textit{if } A \textit{ or } B \textit{ is empty} \\ 1 + LCSS_{\delta,\varepsilon}(Head(A_x), Head(B_x)), & \textit{if } |a_{x,n} - b_{x,m}| < \varepsilon \textit{ and } |n - m| \leq \delta \\ \max(LCSS_{\delta,\varepsilon}(Head(A_x), B_x), LCSS_{\delta,\varepsilon}(A_x, Head(B_x))), & \textit{otherwise} \end{cases}$$

(4.2)

where δ is a constant that controls how far we can look in the past and ε is a constant that controls the size of proximity in which we are looking for matches. $LCSS_{\delta,\varepsilon}(A_y, B_y)$ is defined in a similar way.

Definition 4.2: We define the similarity function $S1_{2D}(\delta, \varepsilon, A, B)$ between two trajectories A and B, given δ and ε as follows:

$$S1_{2D}(\delta, \varepsilon, A, B) = \frac{LCSS_{2D}(\delta, \varepsilon, \rho, A, B)}{\min(n, m)} = \left(\frac{LCSS_{\delta,\varepsilon}(A_x, B_x)}{\min(n, m)}, \frac{LCSS_{\delta,\varepsilon}(A_y, B_y)}{\min(n, m)} \right) \quad 4.3$$

This function is used later to define another similarity measure that will handle parallel trajectories. Another element that we will need in defining the new measure is the family of translations F. A translation is represented as a couple (c_x, c_y), where the components are the values of displacement in each dimension.

$$F = \{ f_{c_x,c_y} \mid f_{c_x,c_y}(A) = ((a_{x,1} + c_x, a_{y,1} + c_y), \ldots, (a_{x,n} + c_x, a_{y,n} + c_y)) \}, \quad 4.4$$

where c_x and c_y are real numbers and A is an arbitrary data sequence. We define also the translation of a time sequence projection on the x-axis:

$$f_{c_x}(A_x) = ((a_{x,1} + c_x),\ldots,(a_{x,n} + c_x)) \qquad 4.5$$

Definition 4.3: Given δ, ε and the family F of translations we define the similarity functions $S2_{2D}(\delta, \varepsilon, A, B)$ between two trajectories A and B as follows:

$$S2_{2D}(\delta,\varepsilon,A,B) = \max_{f_{c_x,c_y}\in F} S1_{2D}(\delta,\varepsilon,A,f_{c_x,c_y}(B)) \qquad 4.6$$

The similarity measure $S2_{2D}(\delta, \varepsilon, A, B)$ is an obvious enhancement of $S1_{2D}(\delta, \varepsilon, A, B)$. $S2_{2D}(\delta, \varepsilon, A, B)$ is able to compute the similarity between sequences that are in different space regions and even if the sequences are in the same region, the way of computing $S1_{2D}(\delta, \varepsilon, A, B)$ does not guarantee that we will get the best match of the two trajectories.

Using $S1_{2D}(\delta, \varepsilon, A, B)$ and $S2_{2D}(\delta, \varepsilon, A, B)$ we can define now the distances between trajectories:

Definition 4.4: Given δ and ε we define the following distance functions:

$$D1_{2D}(\delta,\varepsilon,A,B) = \frac{1}{S1_{2D}(\delta,\varepsilon,A,B)} \text{ and } D2_{2D}(\delta,\varepsilon,A,B) = \frac{1}{S2_{2D}(\delta,\varepsilon,A,B)} \qquad 4.7$$

Regarding the properties of the defined functions it is worth mentioning that $S1_{2D}(\delta, \varepsilon, A, B)$ and $S2_{2D}(\delta, \varepsilon, A, B)$ range from 0 to 1. Therefore $D1_{2D}(\delta, \varepsilon, A, B)$ and $D2_{2D}(\delta, \varepsilon, A, B)$ will range from ∞ to 1. Also, both $D1_{2D}(\delta, \varepsilon, A, B)$ and $D2_{2D}(\delta, \varepsilon, A, B)$ are symmetric functions because $LCSS_{2D}(\delta, \varepsilon, A, B) = LCSS_{2D}(\delta, \varepsilon, B, A)$ and translation is a transformation that preserves symmetry.

4.4 Efficient Algorithms for Computing Similarity

This section will describe a way of increasing the speed of computing the similarity functions $S1_{2D}(\delta, \varepsilon, A, B)$ and $S2_{2D}(\delta, \varepsilon, A, B)$.

4.4.1 Computing the similarity function S1

The function $S1_{2D}(\delta, \varepsilon, A, B)$ computes the similarity between sequences A and B by applying the LCSS algorithm. The running time of this algorithm (described in [30]) is $O(m \cdot n)$. Unlike the regular LCSS algorithm we have the restriction that we cannot match points that have a time gap between them bigger than δ. This restriction will allow us to increase the speed of computation. The following lemma is proved in the appendix:

Lemma 4.1: Given two trajectories *A* and *B*, with lengths *n* and *m* respectively, we can find the $LCSS_{2D}(\delta, \varepsilon, \rho, A, B)$ in $O(\delta\cdot(n+m))$ time.

4.4.2 Computing the similarity function S2

Function $S2_{2D}(\delta, \varepsilon, A, B)$ computes the similarity between sequences *A* and *B* by finding the translation f_{c_x,c_y} that maximizes the length of the longest common subsequence of *A* and $f_{c_x,c_y}(B)$ over the set of all possible translations. Even though there the number of possible translations is infinite, we can determine a set of significant transformations and use only these in the process of computing $S2_{2D}(\delta, \varepsilon, A, B)$. This method, of reducing the size of translation set, was presented in [18] and [16].

Let *A* and *B* be two time sequences whose lengths are *n* and *m* respectively. We can represent their projections A_x and B_x, in a plane, where elements of A_x are on the y-axis and elements of B_x on the x-axis (see figure 4.1 below).

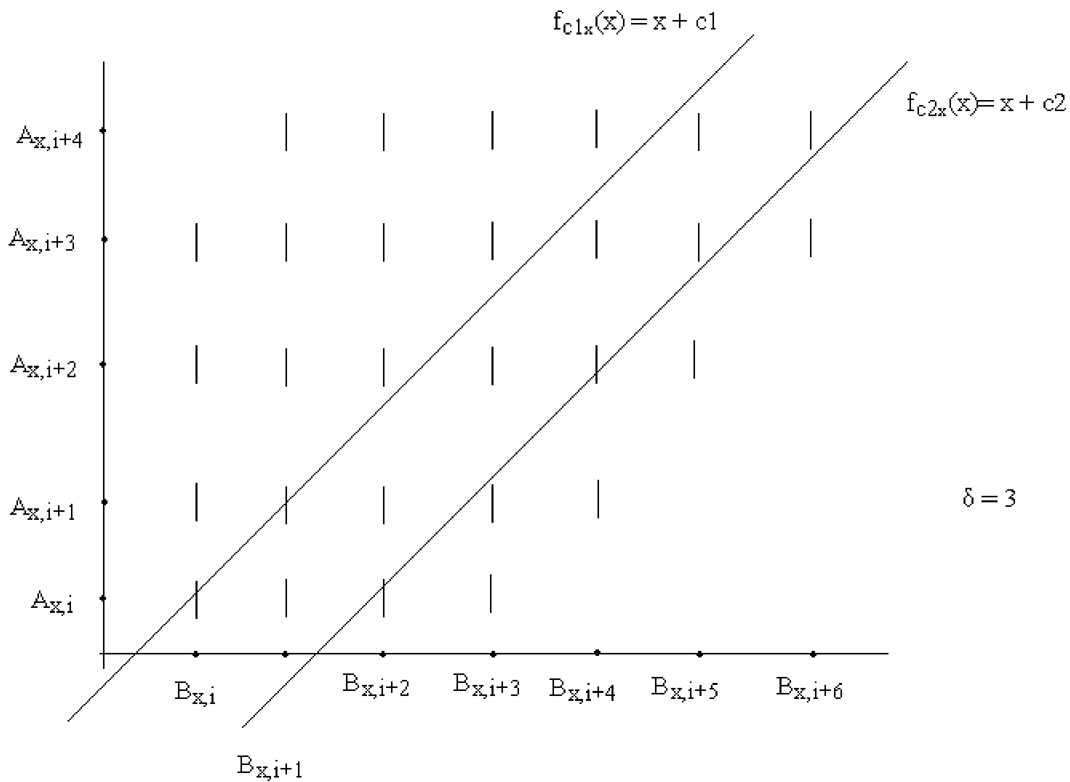


Fig. 4.1 Finding the translations between two trajectories

Translating a projection by the amount *c* is represented by the linear transformation $f_c(b_{x,i}) = b_{x,i} + c$. This transformation will allow matching $b_{x,i}$ to all $a_{x,j}$ for which the following conditions hold: $|i - j| < \delta$ and $a_{x,j} - \varepsilon \leq f(b_{x,i}) \leq a_{x,j} + \varepsilon$.

Matching conditions of elements are represented in Fig 4.1 as small vertical segments of length $2\cdot\varepsilon$ associated with every pair $(b_{x,i}, a_{x,j})$ and centered in $(b_{x,i}, a_{x,j})$. Since each element of A_x can be matched with at most $2\cdot\delta + 1$ elements of B_x it follows that the total number of this kind of segments is $O(\delta\cdot n)$.
In the above representation, translating B_x by the quantity c and then matching its elements against the elements of A_x is done by drawing the line $f_c(b_{x,i}) = b_{x,i} + c$ and counting how many line segments $((b_{x,i}, a_{x,j} - \varepsilon), (b_{x,i}, a_{x,j} + \varepsilon))$ it intersects. The number of intersections is an upper bound of the length of the longest common subsequence (sometimes the line $f_c(b_{x,i})$ might intersect two segments associated with the same $a_{x,j}$). Two different translations will generate two different longest common subsequences only if they will intersect two different sets of segments. Consequently, we can divide the set of all translations into equivalence classes. A class will be composed of all translations that intersect the same line segments. In addition, there will be a class consisting of translations that do not intersect any line segments. What we will need in our computation of function $S2_{2D}(\delta, \varepsilon, A, B)$ is a representative of each equivalence class (except the last one, of translations that do not intersect any line segment). An upper bound on the number of possible longest common subsequences is given in the following lemma presented in [16]:

Lemma 4.2: Given two one dimensional sequences A_x and B_x, there are $O(\delta\cdot(m+n))$ lines of slope 1 that intersect different sets of line segments.

Proof: Let us consider the line given by the function $f_c(x)= x + c$. This line has slope 1, and it will intersect a certain number of line segments. We can translate it to the left or to the right and as long as it intersects the same line segments, the line will stay in the same equivalence class. As soon as one of the line segment endpoints is crossed, the line will move in a different equivalence class. There are $O(\delta\cdot(m+n))$ endpoints so a line that sweeps all the points will intersect $O(\delta\cdot(m+n))$ equivalence classes during the sweep.

We can compute these $O(\delta\cdot(m+n))$ translations (representatives for each equivalence class) by finding the lines that cross the endpoints of each line segment. This set of translations, of size $O(\delta\cdot(m+n))$, will provide all possible matchings for computing the longest common subsequence between A_x and B_x. A similar routine is used for finding the corresponding translation set for A_y and B_y. Because we compute these sets independently of each other and the longest common subsequence between A and B is computed as a vector (D_x, D_y), the necessary time to finish this computation is $O(\delta\cdot(m+n))$. In addition, the running time of the LCSS algorithm is $O(\delta\cdot(m+n))$. We can summarize these in the following theorem, similar to the one presented in [16]:

Theorem 4.1: Given two trajectories A and B, whose lengths are n and m respectively, we can compute the $S2_{2D}(\delta, \varepsilon, A, B)$ in $O(\delta^2\cdot(m+n)^2)$ time.

4.5 Clustering

Clustering trajectories is the last step of trajectory processing. Developing methods to classify (cluster) objects according to perceived similarities is an important research area, with application in many sciences, including computer vision. In the field of clustering trajectories, there is a significant amount of past and ongoing research for defining good similarity measures that will lead to high-quality clusters of trajectories based on some of their features.

There are several algorithms used for computing the clusters given a set of data. For our project we use a modified version of the classic algorithm for "Agglomerative Hierarchical Clustering" presented by Duda et al. in [14]. We chose this algorithm because the data that we get from previous stage is a set of 2D vectors representing the distance between each pair of trajectories. Unlike the algorithm presented in [14], we have an extra problem to deal with: the number of clusters, which is unknown. Agglomerative hierarchical clustering addresses this problem by building a dendrogram for our set of data and analyzing the results at the end of the computation. Also, in certain cases we can get an infinite distance between trajectories. These trajectories will be considered from the beginning as being in different clusters. Therefore in our approach final number of clusters will be either one or minimum number of clusters that will satisfy the infinite distance criteria (any two trajectories that have infinite distance between them, will be placed in different clusters).

The algorithm for finding the clusters is presented below. c_{final} represents the final number of clusters and its value is computed as described above, v_{i} represents the trajectory *I*, D_{j} cluster *j* and *n* is the total number of trajectories.

Algorithm 4.1: Agglomerative Hierarchical Clustering

1. c_{final}, $c_{\text{init}} \leftarrow n$, $D_{\text{i}} \leftarrow \{v_{\text{i}}\}$, $i = 1, \ldots, n$
2. **if** $c_{\text{init}} = c_{\text{final}}$
3. **then return** c_{init} clusters
4. **do** $c_{\text{init}} \leftarrow c_{\text{init}} - 1$
5. find nearest clusters, say D_{i} and D_{j}
6. merge D_{i} and D_{j}
7. **while** $c_{\text{final}} \neq c_{\text{init}}$
8. **return** c_{init} clusters

The above algorithm ends when the desired number of clusters was reached. An important part of it that was not mentioned above is how we compute the distance between clusters. In our approach we tried several distance definitions:

$$d_{\min}(D_i, D_j) = \min_{A \in D_i,\, B \in D_j} \left\| D2_{2D}(\delta, \varepsilon, A, B) \right\| \quad 4.8$$

$$d_{\max}(D_i, D_j) = \max_{A \in D_i,\, B \in D_j} \left\| D2_{2D}(\delta, \varepsilon, A, B) \right\| \quad 4.9$$

$$d_{avg}(D_i, D_j) = \frac{1}{n_i \cdot n_j} \cdot \sum_{A \in D_i} \sum_{B \in D_j} \left\| D2_{2D}(\delta, \varepsilon, A, B) \right\| \quad 4.10$$

These measures perform well and yield the same results if the clusters are compact and well separated. Nevertheless, if clusters are close to one another, or if their shapes are not basically hyperspherical, the results that we get might be different.

When $d_{\min}(\cdot,\ \cdot)$ is used to measure the distance between clusters (Eq. 4.8) the algorithm is sometimes called the *nearest-neighbor cluster algorithm* or *minimum algorithm*. In addition, if a threshold is used to stop the algorithm when the distance between clusters is above the threshold, it is called *single-linkage algorithm*. A disadvantage of this distance is that in some cases it may produce an elongated cluster (behavior named “chaining effect”).

A similar distance, $d_{\max}(\cdot,\ \cdot)$ (Eq. 4.9) is used to measure the distance between subsets. The algorithm that is using this distance is called the *farthest-neighbor algorithm* or *maximum algorithm*. When a threshold is used to limit size of clusters, the algorithm is called *complete-linkage algorithm*. This algorithm discourages the development of elongated clusters. However, there are cases when using this distance will yield meaningless results.

These two distances are very sensitive to outliers. Hence, a compromise for improving this problem is the third distance $d_{\text{avg}}(\cdot,\ \cdot)$ (Eq. 4.10). Moreover, we can use $d_{\text{avg}}(\cdot,\ \cdot)$ in any algorithm where we used the other two distances and complexity of computing $d_{\text{avg}}(\cdot, \cdot)$ is similar to the one of computing $d_{\min}(\cdot, \cdot)$ and $d_{\max}(\cdot, \cdot)$.

Computational complexity of these distances is discussed in [14]. Considering that we have n patterns, in a d-dimensional space and we want to find c_{final} clusters using one of the distances defined in Eqs. 4.8-4.10. At the beginning we will need to compute $n \cdot (n - 1)$ inter-point distances, each one of complexity $O(d)$. Finding the minimum, maximum or average distance pair (for the first merging) requires that we browse the whole list of distances keeping track of the minimum and maximum distance respectively (while for $d_{\text{avg}}(\cdot,\ \cdot)$ we do not need to do anything here). Hence, the complexity of first agglomerative step is $O(n^2 d)$. For an arbitrary agglomeration step (from c_{init} to $c_{\text{init}} - 1$), an estimation of number of distances that we need to browse is given by $n \cdot (n - 1) - (n - c_{\text{init}}) \cdot (n - c_{\text{init}} - 1) = c_{\text{init}}^2 + 2 \cdot n \cdot c_{\text{init}} - c_{\text{init}}$. So, the complexity is $O(c_{\text{init}}^2 + n \cdot c_{\text{init}})$. Thus, the full time complexity is $O((n - c_{\text{init}}) \cdot n^2 \cdot d)$.

Chapter 5

Experiments

The system was implemented and tested using a dual-processor Windows machine, (each processor with 1 GHz frequency) with one GB of memory and a Matrox capture card. We tested our system using real and synthetic data. Real data was either a live stream video, captured using a Sony camera or a prerecorded video sequence stored on tape or in a movie file.

5.1 Estimation and Prediction of Trajectories

The following example shows the result of tracking moving objects in a video sequence. This video sequence was taken from the website of PETS 2001 (Performance Evaluation of Tracking and Surveillance Workshop) [35].

Fig 5.2 shows 6 pairs of images (six frames extracted from the video clip together with the corresponding segmented frames that show the extracted blobs with their trackers). Pairs 2, 3 and 4 show tracker behavior during occlusion. The values of matrix R_k in the EKF estimation were changed such that the tracker will rely mostly on previous estimations of the position and measurement update of the position will have a diminished influence in computing the actual estimation. Once the occlusion disappears the values in matrix R_k were changed back. Pairs 5 and 6 show how the trackers will retrieve their corresponding blob. Finally, the trajectories of the two blobs are shown below in Fig 5.1

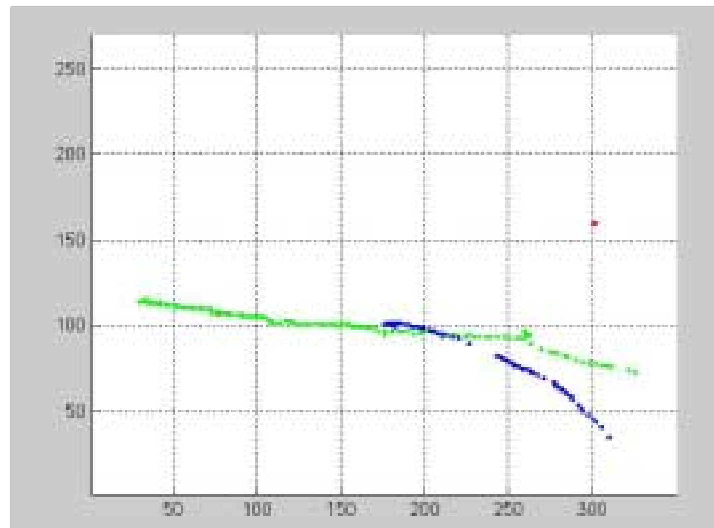


Fig 5.1

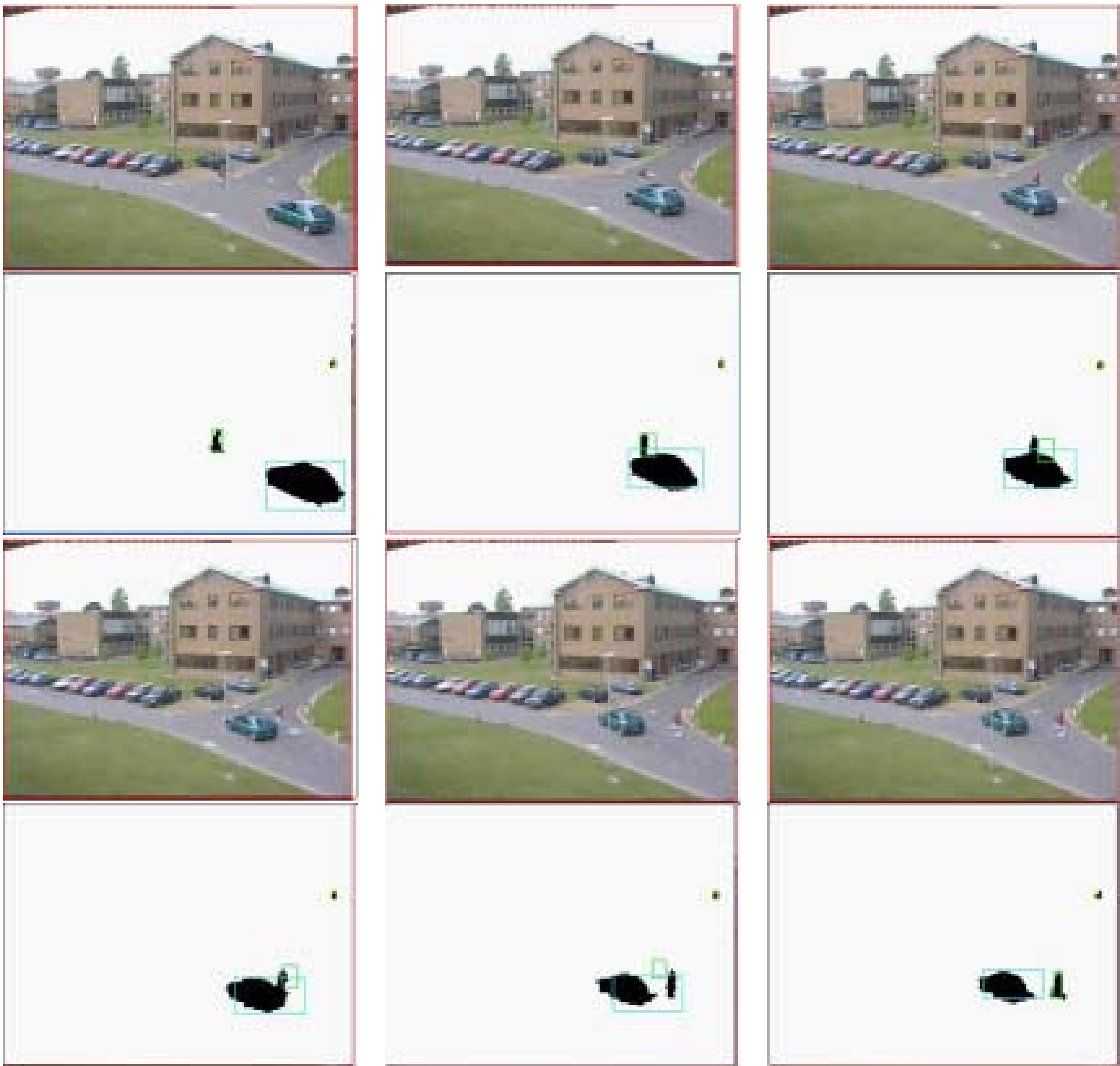
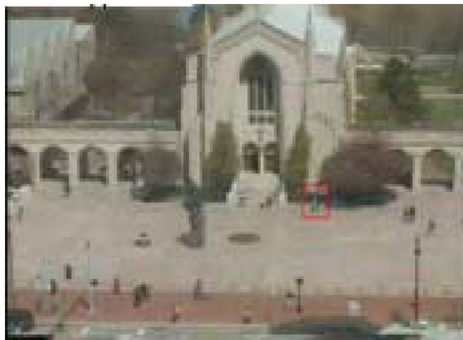


Fig 5.2

The following example shows how we can improve the trajectory estimation in space by using a homography to eliminate projection distortion. This sequence ("square" video sequence) shows a person walking through a square and entering a building. Every frame is accompanied by a projection of the ground plane showing the trajectory transformed using the defined homography.



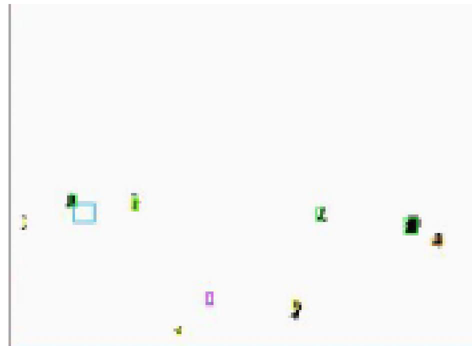


Fig 5.3

Nevertheless, our EKF implementation has its weaknesses. For example, if during the occlusion there are four or five blobs overlapped, some trackers might lose their object and need to be destroyed. Also, even if in the occlusion there are only two blobs, if the trackers are not stable enough they may lose the trace of their objects.

Another case is when the object temporary disappears from the scene, either because is occluded be the elements of the background or because the object color is very similar to the one of the background and the system is not able to segment it correctly. Our system allows an object to disappear from the scene for three consecutive processed frames. We needed to find a balance between destroying a tracker to early (and creating a new one for the same blob) and destroying it too late (and allowing other blobs to be associated with our tracker). The following example, taken from PETS website [35], shows how the tracker is lost when the object is occluded by the background.

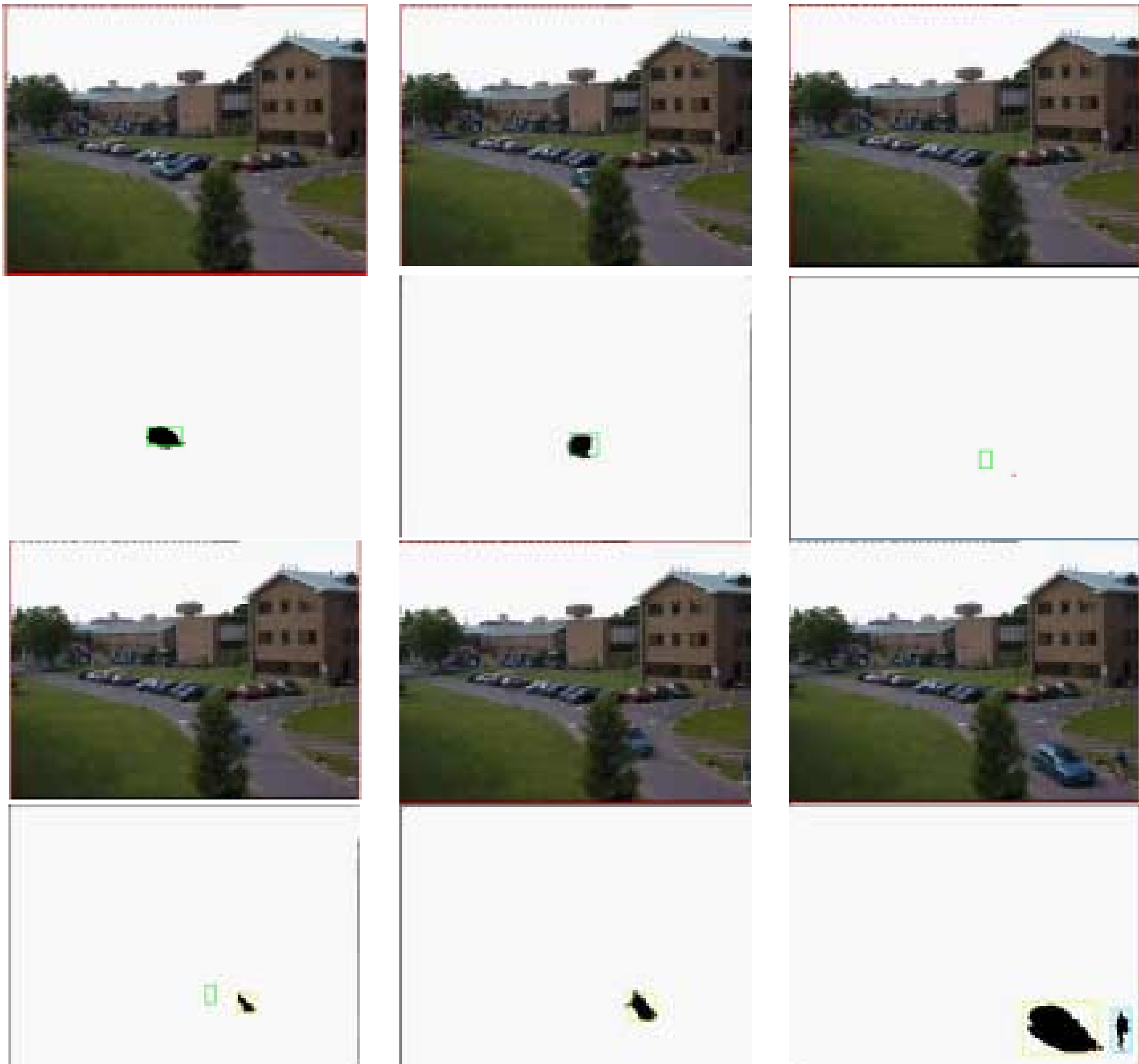


Fig 5.5

Following images shows the trajectories extracted from several datasets. Figure 5.6 shows the trajectories extracted from the dataset presented at the beginning of the chapter. Figure 5.7 shows the trajectories extracted from previous dataset. It is obvious the presence in the scene of an obstruction that breaks the trajectories in the middle.

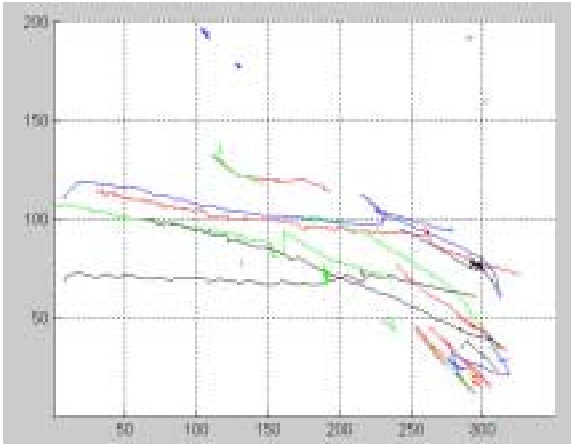


Fig 5.6

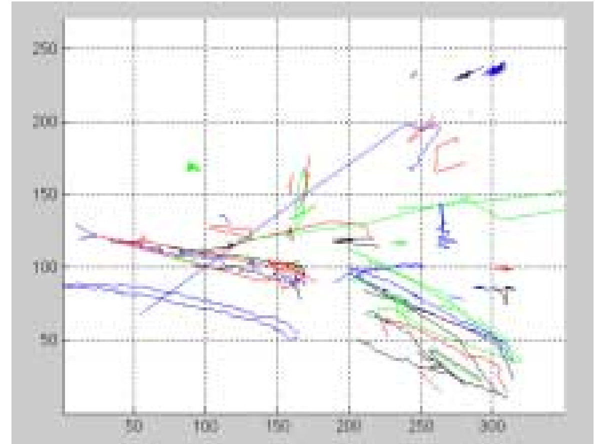


Fig 5.7

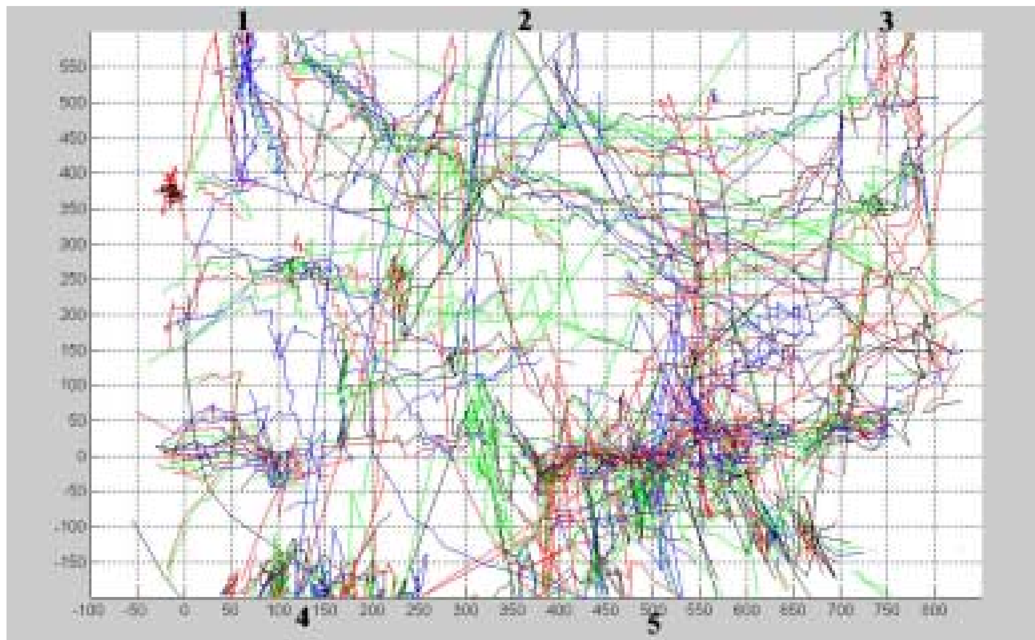


Fig 5.8

1 – left arcades
2 – church entrance
3 – right arcades
4 – street crossing
5 – street crossing

Fig 5.9

Fig 5.8 shows a map of trajectories extracted from the ”square” video sequence. Projection distortion of these trajectories was removed by applying a linear transformation, a homography. Fig 5.9 shows an image of the square as seen on the view plane and numbers on Fig. 5.8 and 5.9 show the corresponding areas in both images.

5.2 Computing Similarities and Clustering Trajectories

Using the trajectories extracted in the previous stage, we computed the similarities between them. Further, we ran our clustering algorithm using these similarities to group trajectories together.

In computing the similarities between time sequences we choose as minimum ratio between two trajectories $\rho = 0.4$, where this ratio is defined as the ratio between the length of the shorter sequence and the length of the longer sequence. Reason for choosing this value was to avoid comparison of real trajectories with false trajectories created by noise, which in general are short. The other parameter, δ (how much we can go back in time when comparing the trajectory) was chosen 0.5, meaning that we can go back in time as much as half of the length of the shorter trajectory. For the last parameter, ε, the one that controls the spatial matching, we tested our algorithm using three different values 3, 10, 40.

Finally, the clustering algorithm, as was described in Section 4.4 was tested using three different types of distances between clusters: minimum distance (Eq. 4.8), maximum distance (Eq. 4.9) and average distance (Eq. 4.10). The norm used for vectors (we computed distances between trajectories as pairs of two real numbers) involved in computing distances between clusters was chosen to be Euclidean norm.

Similarities that we previously computed will reflect how similar are our trajectories. Unfortunately, the inter-trajectories distance that we defined using the similarity is not a metric so we need to be careful how we define our clusters. A natural requirement will be that if the distance between two trajectories has a big value (the value chosen for distances when similarity is zero), then the two trajectories should be in different clusters. The second requirement is that the distances between sequences existing in a cluster will be smaller than the distance of an outside trajectory to the cluster. The reason for the first requirement started when we defined similarity between two sequences. If the similarity between two sequences is zero, the distance between them will be infinite. It seems natural to place them in different clusters. When the similarity between two sequences will be zero? When the ration between the length of the shorter sequence and the length of the longer sequence will be smaller than the minimum ratio. Reason for enforcing this restriction is to avoid getting false results. Without this restriction, for example, a trajectory of only two points will have a high degree of similarity with any trajectory, regardless the latter trajectory length. The second requirement is a natural one too. Considering that similarity is inverse proportional with the distance, the bigger the distance between trajectories, the smaller the similarity.

First example is the result for computing similarities for a set of trajectories and for computing the clusters of these trajectories using the three distances presented above. Set of data is a synthetic one, with four clusters shown in Fig 5.10 representing trajectories of 10 points. Clustering results are shown in Fig 5.11. Left column represents the results of maximum distance algorithm (with ε being 3, 10, 40), while right column represents the results of average distance algorithm (with ε being 3, 10, 40). Results for single-linkage algorithm were not considered relevant in this case and are not shown.

Analyzing Fig 5.11 we can see that number of clusters depends not only on inter-cluster distances but also on value of ε (size of spatial matching). When value of ε is 3 we hardly see the interval of inter-cluster distances for which we have 4 clusters. As the size of the matching interval increases, the size of interval of inter-cluster distances for which we have 4 clusters increases too.

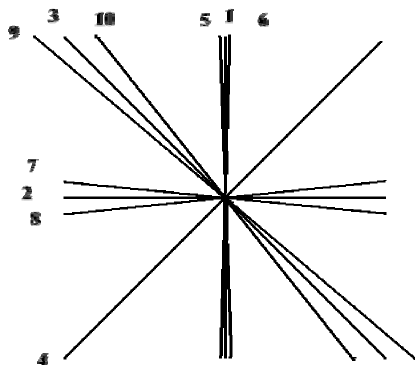


Fig 5.10

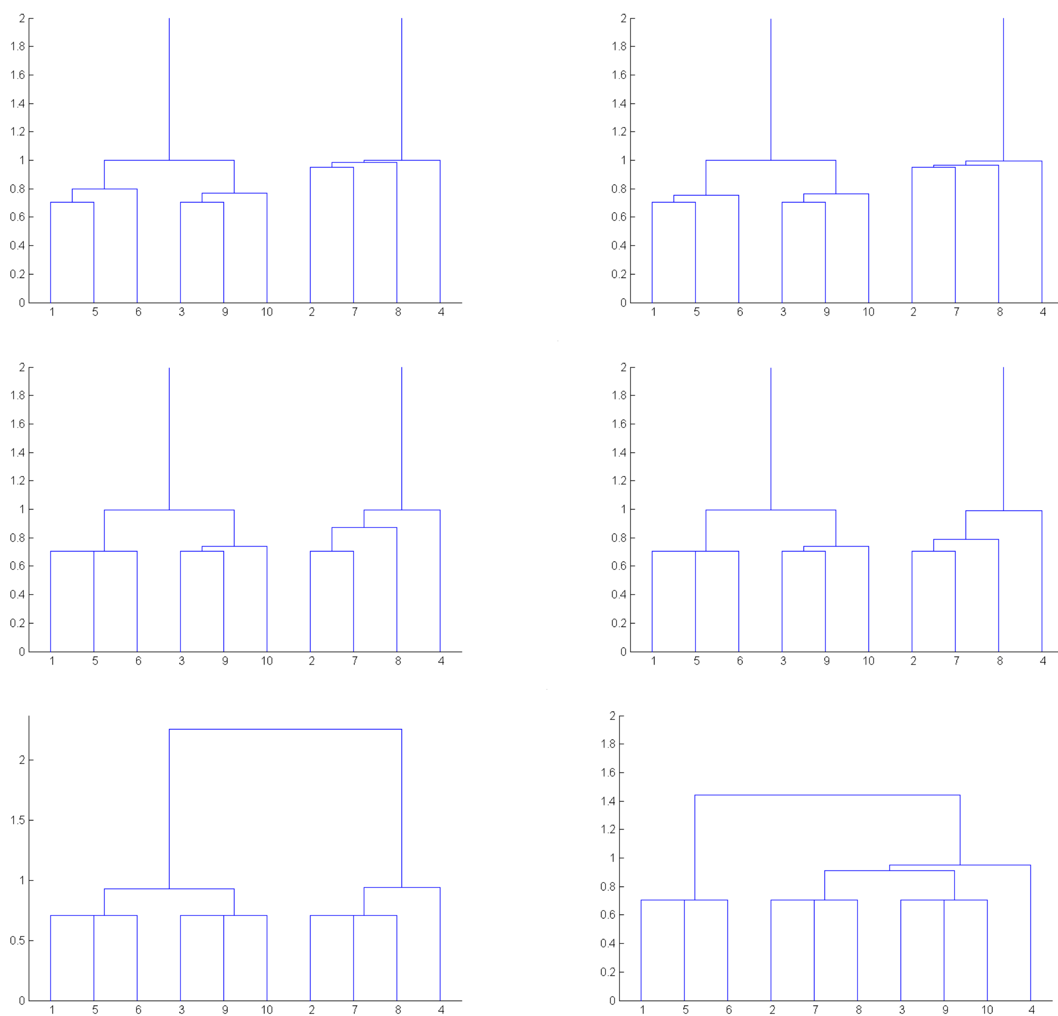


Fig 5.11
Clusters for synthetic data from Fig 5.10
with $\varepsilon = 3$ (first row), $\varepsilon = 10$ (second row), $\varepsilon = 40$ (third row)

Nevertheless, if ε is too big, the size of this interval will eventually become zero. A similar fact can be seen in Fig 5.11. In the second row trajectory groups 3, 9, 10 and 2, 7, 8 are part of two different clusters: ((3,9) 10) and ((2, 7), 8), while in the third row each group will form a single cluster: (3, 9, 10) and (2, 7, 8).

Next, we will present here the results for computing similarities for a set of trajectories and for computing the clusters of these trajectories using the three distances presented above. The initial set of data, the similarities between each pair of trajectories, is hard to represent because it represents relative distances between each pair of trajectories and moreover, the distance that we defined is not a metric because it does not obey the triangle inequality. We used multidimensional scaling to decrease dimensionality of our set of data and then represent the first three columns of the matrix that embeds our set of data in the lower dimension space. Unfortunately, the approximation errors are big, so the results that we got are a poor approximation of the initial set of data.

For the first set of trajectories (Fig 5.6) the result of applying of the above method (multidimensional scaling) is shown in the Figure 5.12 (regardless of the value of parameter ε):

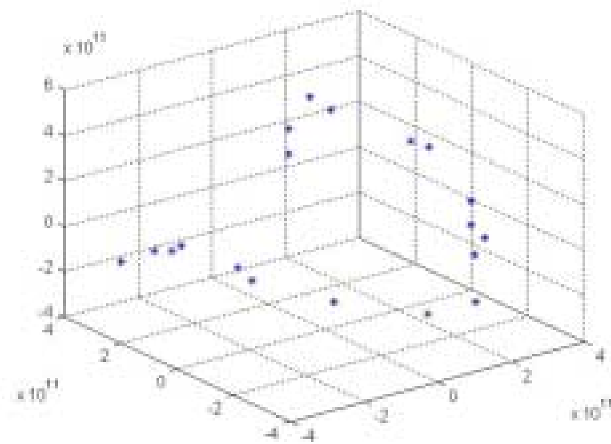


Fig 5.12

For this set of trajectories, ground truth is unknown. In some cases, even representing the trajectories in 3D (the third axis being time axis) will not help us to see the clusters because the objects in our scene might have different velocities. Nonetheless, we can determine minimum number of clusters expected by analyzing initial set of distances between our trajectories. The set of trajectories will be divided between several subsets that will obey the first requirement. In our case we will get 8 subsets. Therefore, we will expect at least eight clusters and those clusters will have to obey the second requirement too.

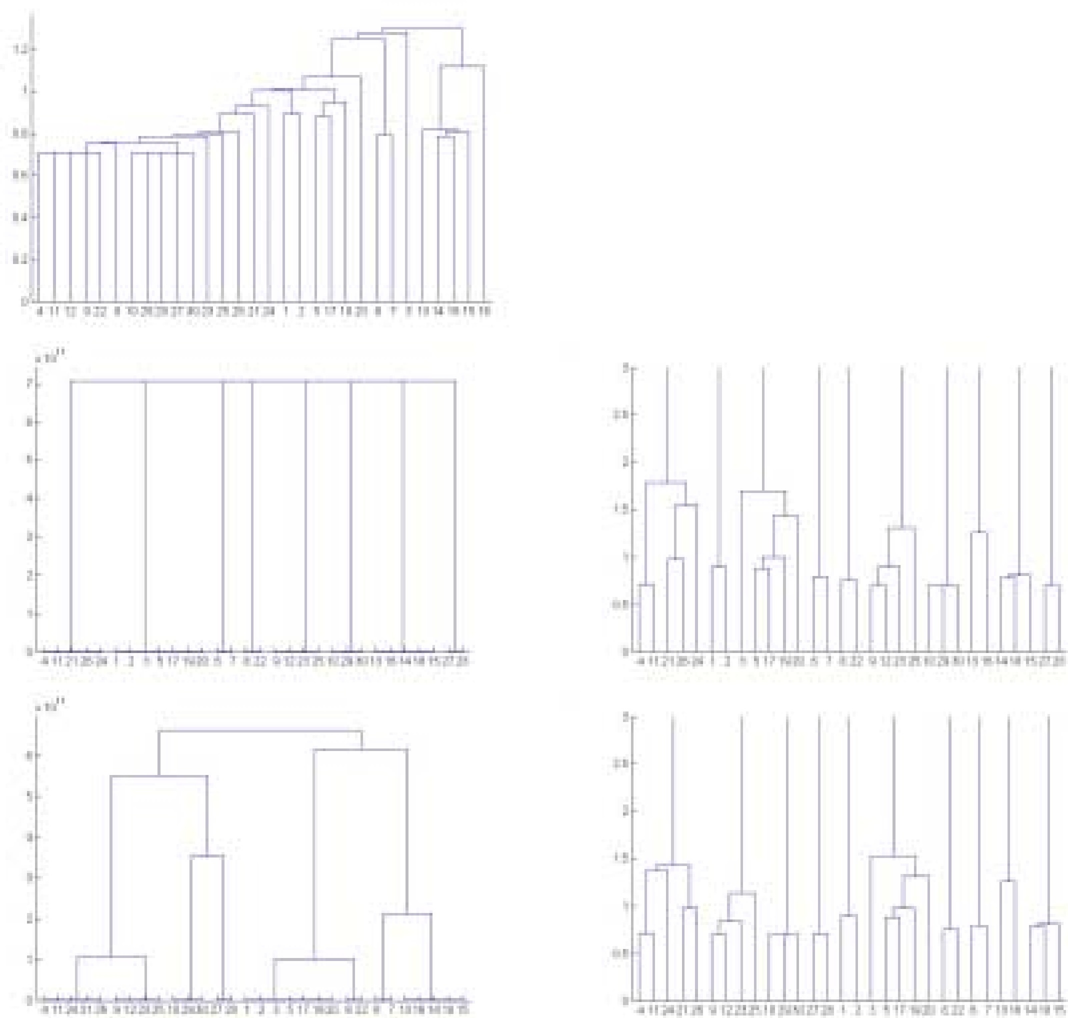


Fig 5.13

Clusters resulted when $\varepsilon = 3$ (left column unscaled, right column scaled)
(first row – minimum distance, second row – maximum distance, third row average distance)

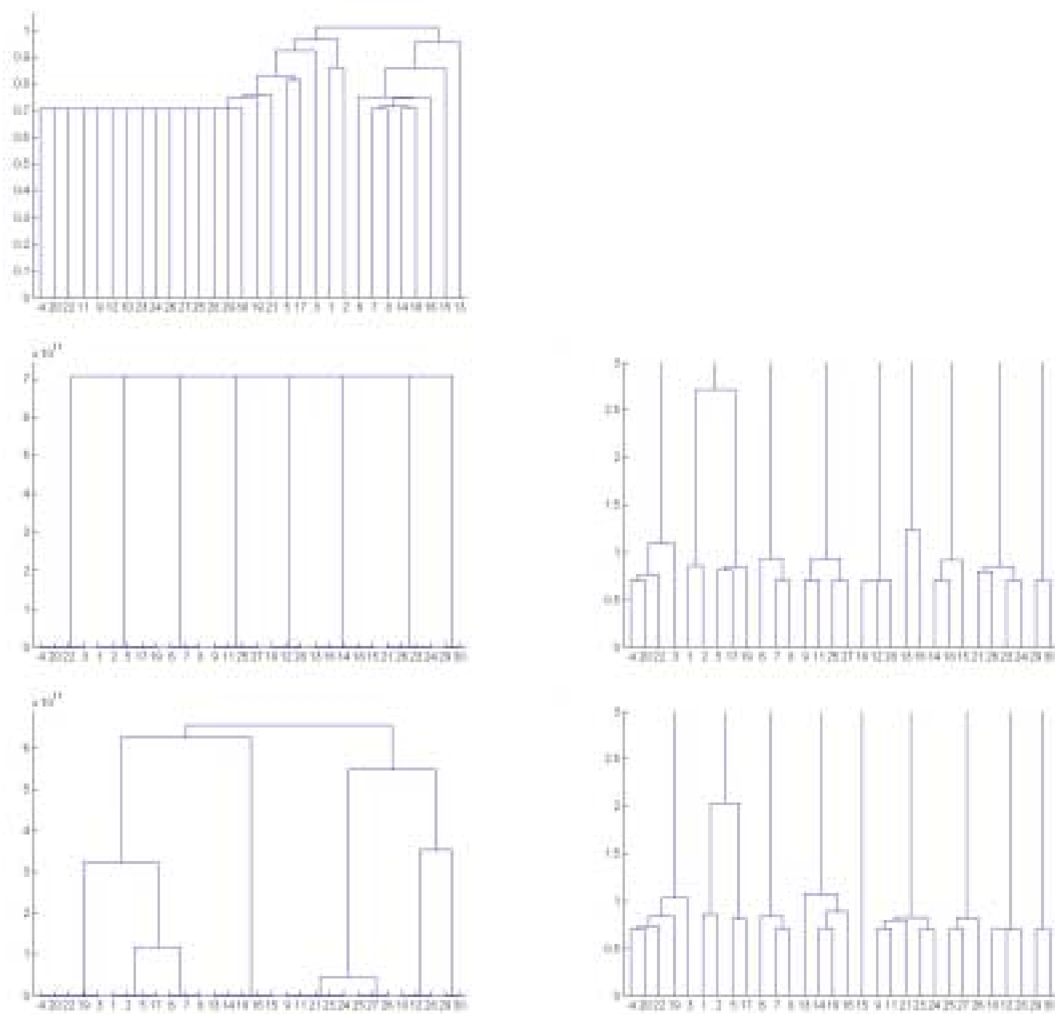


Fig 5.14
Clusters resulted when $\varepsilon = 10$ (left column unscaled, right column scaled)
(first row – minimum distance, second row – maximum distance, third row average distance)

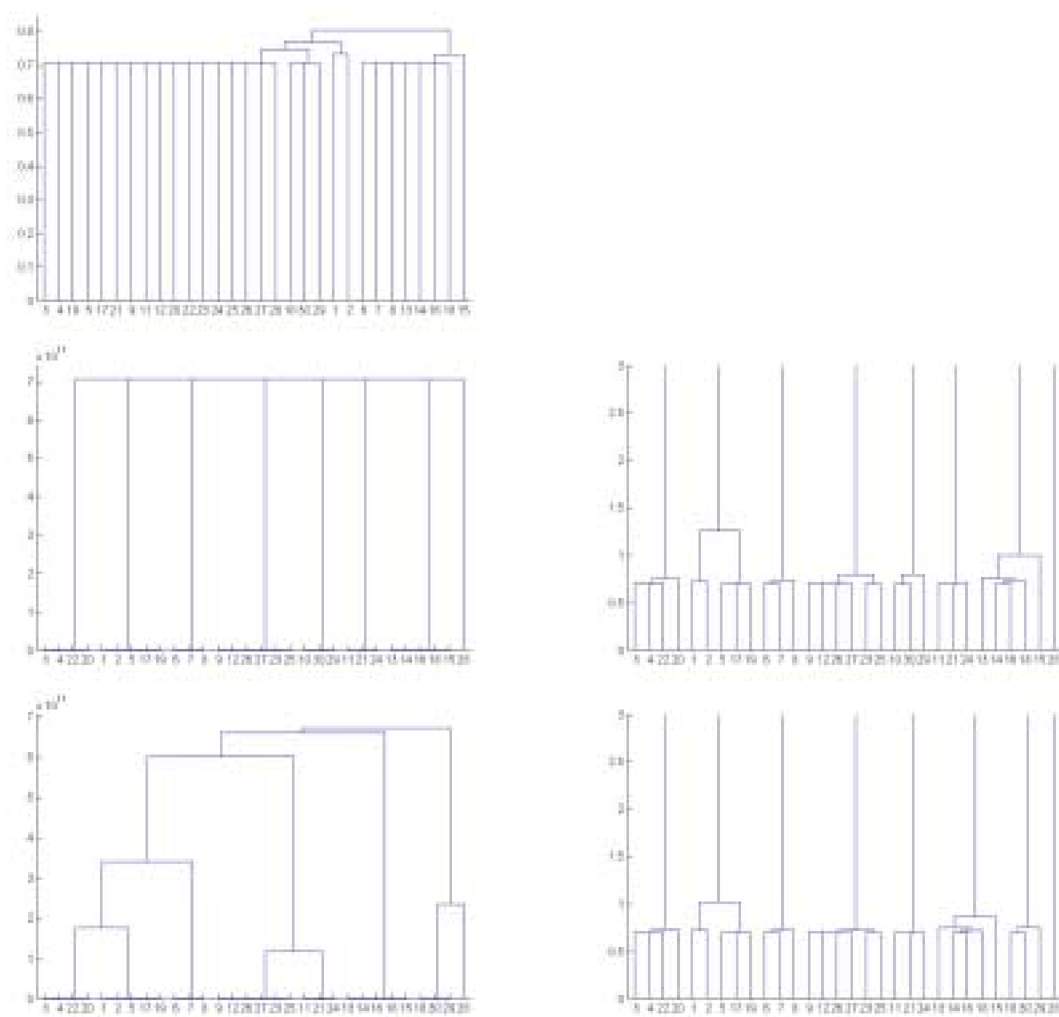


Fig 5.15
Clusters resulted when $\varepsilon = 40$ (left column unscaled, right column scaled)
(first row – minimum distance, second row – maximum distance, third row average distance)

As expected, clustering algorithm that uses minimum distance did not perform well. A good example is in the first row image of Fig 5.13, where trajectories 4, 11, 12, 9 and 22 were grouped in a cluster.

	4	11	12	9	22
4	0	0.707107	∞	1.06902	0.806223
11		0	0.707107	0.74257	0.707107
12			0	0.707107	∞
9				0	∞
22					0

Table 5.1

Graphical representation of this cluster in 3D shows the differences in size of these trajectories (in the right image of Fig 5.16 x-axis is the time axis):

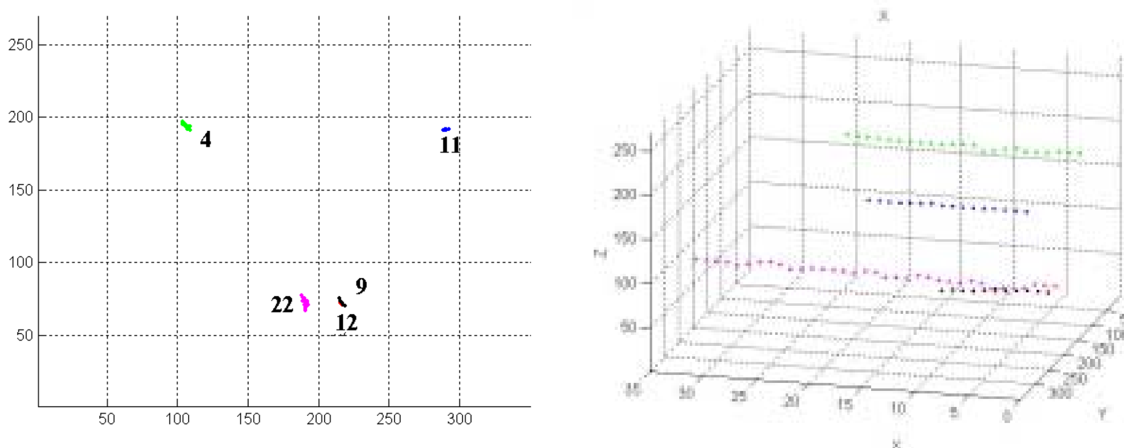


Fig 5.16

Moreover, studying the dendrogram of Fig 5.13 – 5.15 for single–linkage algorithm, the limit for eight clusters is not as clear visible as it is for maximum distance and average distance. For single–linkage algorithm whose dendrogram is presented in 5.15, this limit don't even exist.

For maximum distance and average distance algorithms, the minimum number of eight clusters that we asked for is clearly visible in all Fig 5.13 – 5.15. Studying the second and third row of images in Fig 5.13 – 5.15 we can see that the results provided by the clustering algorithms that use maximum distance and average distance are almost identical from the point of view of trajectories that are included in a cluster. Another difference is the magnitude of distances that separates clusters. For example, for Fig 5.13 if the distance between clusters is in the interval [9, 36] maximum distance will generate 9 clusters while average distance will generate 8 clusters.

More significant differences in cluster structure occur when we increase the size of potential matching area.

ε = 3	4,11, 21,26, 24	1,2,3,5, 17,19, 20	6,7	8,22	9,12, 23, 25	10,29, 30	13,16, 14,18, 15	27,28
ε = 10	4,20, 22,3	1,2,5, 17,19	6,7,8	21,26, 23,24	9,11, 25,27	10,12, 28	13,16, 14,18, 15	29,30
ε = 40	3,4,22, 20	1,2,5, 17,19	6,7,8	11,21, 24	9,12, 26,27, 23,25	10,30, 29	13,16, 14,18, 15	28

Table 5.2 Clusters for the complete linkage algorithm when distance between clusters is bigger than 40

ε = 3	4,11, 21,26, 24	1,2,3,5, 17,19, 20	6,7	8,22	9,12, 23, 25	10,29, 30	13,16	14,18, 15	27,28
ε = 10	4,20, 22,3	1,2,5, 17,19	6,7,8	21,26, 23,24	9,11, 25,27	10,12, 28	13,16, 14,18, 15		29,30
ε = 40	3,4,22, 20	1,2,5, 17,19	6,7,8	11,21, 24	9,12, 26,27, 23,25	10,30, 29	13,16, 14,18, 15		28

Table 5.3 Clusters for the complete linkage algorithm when distance between clusters is smaller than 39 and bigger than 24

ε =3	4,11, 21,26, 24	1,2	3,5, 17,19, 20	6,7	8,22	9,12, 23,25	10,29, 30	13,16	14,18, 15	27,28
ε=10	21,26, 23,24	1,2,5, 17,19	4,20, 22,3	6,7,8	29,30	9,11, 25,27	10,12, 28	13,16	14,18, 15	
ε =40	11,21, 24	1,2,5, 17,19	3,4, 22,20	6,7,8		9,12, 26,27, 23,25	10,30, 29		13,14, 16,18, 15	28

Table 5.4 Clusters for the complete linkage algorithm when distance between clusters is smaller than 23 and bigger than 3

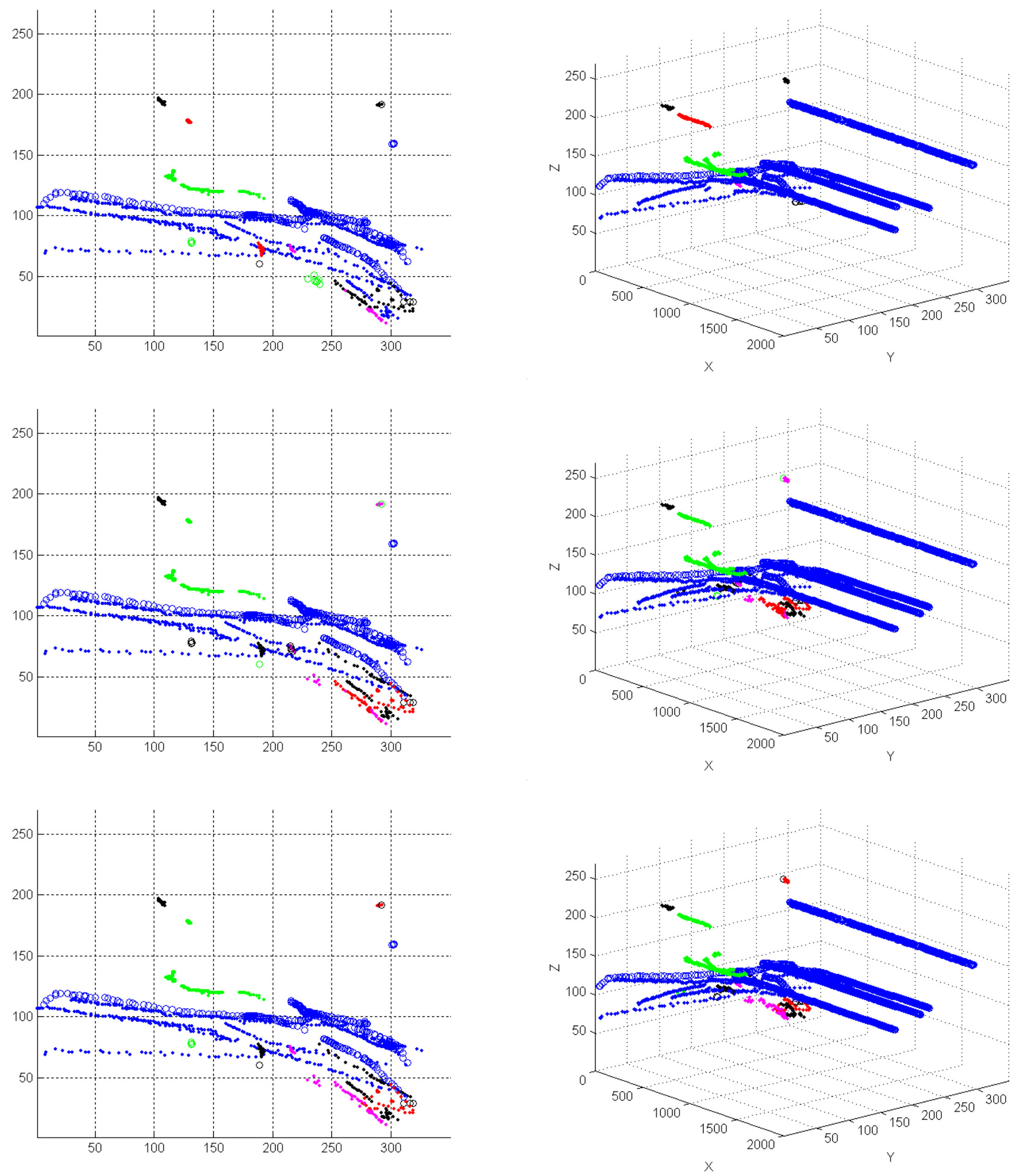


Fig 5.17 Clusters from table 5.2
left column shows 2D representation of trajectories,
right column shows representation of trajectories using x-axis as the time scale
(distance between clusters is bigger than 40)

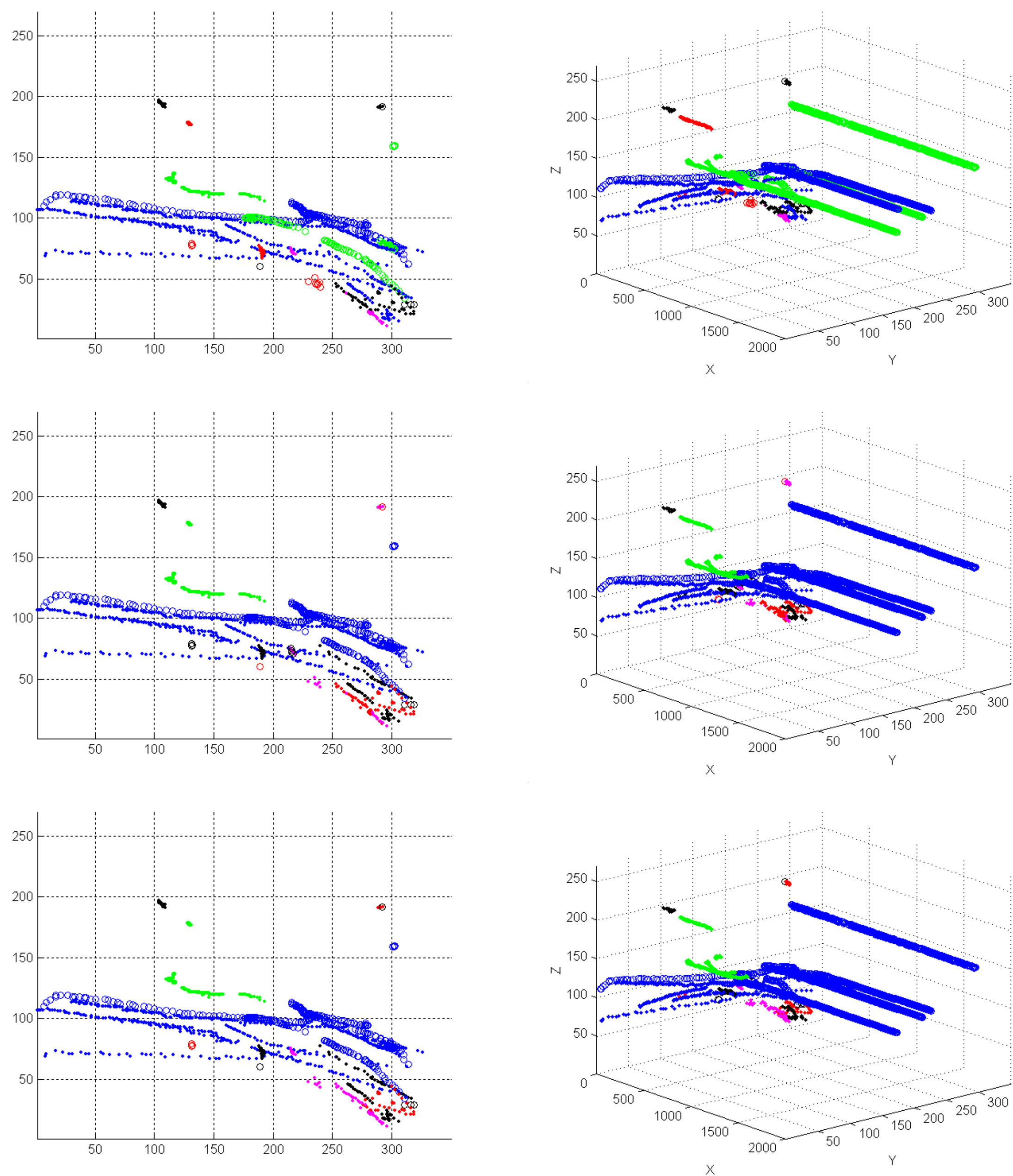


Fig 5.18 Clusters from table 5.3
left column shows 2D representation of trajectories,
right column shows representation of trajectories using x-axis as the time scale
(distance between clusters is smaller than 39 and bigger than 24)

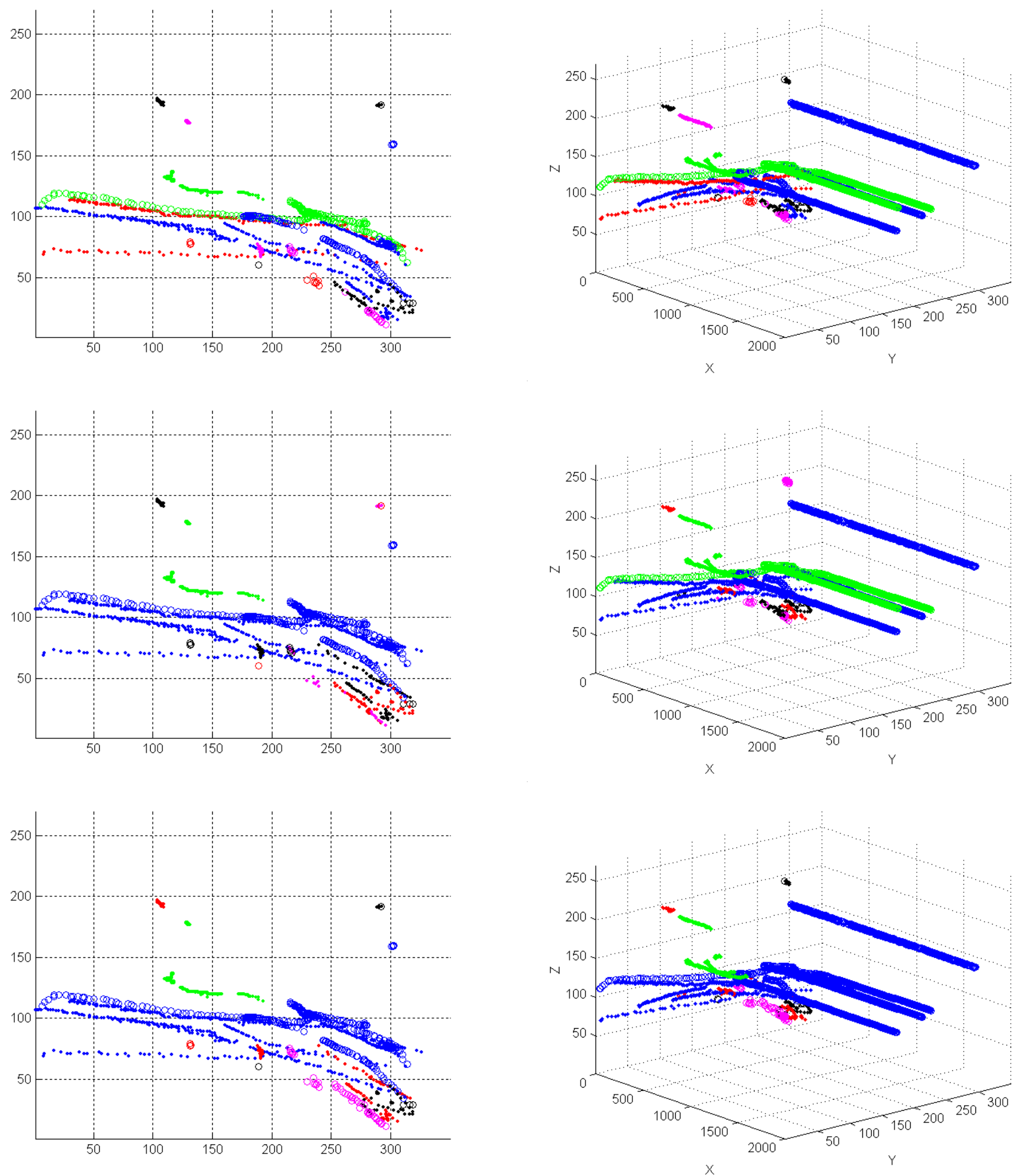


Fig 5.19 Clusters from table 5.4
left column shows 2D representation of trajectories,
right column shows representation of trajectories using x-axis as the time scale
(distance between clusters is smaller than 23 and bigger than 3)

Chapter 6

Conclusions

6.1 Estimation and Prediction of Trajectories

We presented a system for estimating and predicting trajectories of moving objects in a 3D space. Trajectories are considered to be locally linear and motion is assumed to have constant speed. Far from being a drawback of our method, changes in trajectories or acceleration are foreseen in our approach, being included in our EKF formulation. Our system can accurately track multiple objects, even in the case when occlusions occur either between objects or between objects and background. When occlusions disappear our system will try to reattach each tracker to its corresponding object by following multiple hypotheses until a decision can be made. A key element for the success of this operation is stability of the tracker (how good was the estimation of position, trajectory, velocity before the overlapping occurred): the more stable the tracker is, the greater probability of correct reattachment after the occlusion disappeared. To avoid false results, if a decision cannot be made, a new tracker will be created.

To improve the spatial quality of the extracted trajectories, in certain cases (when we have some topographical information of a scene) we are able to remove the projection distortion of these trajectories.

In our system we assume that object depth is zero, and bottom part of the bounding box that borders tracked objects is on the ground plane. Even that the size of the objects is much smaller than the size of the view plane, depending on the scene that we are observing these assumptions can be a problem. For example, if the ground plane is not flat and differences between different levels are pretty big, or if there are obstacles in the ground plane that obstructs the view of the lower part of the moving objects.

Another issue that will affect the quality of estimation of trajectory is the result of segmentation. Because of the noise, shadows, similarity between object and background colors, the size of the object can change quickly from one frame to another. Nevertheless, a stable tracker is able to handle situations like these.

6.2 Computing Similarities and Clustering Trajectories

We described a method for computing the similarities between two time sequences. Having computed the similarities for each pair of trajectories existing in a given set we applied a clustering algorithm for grouping together the trajectories that have common features.

Similarity between two time sequences was defined using the longest common subsequence formulation. We expanded this formulation to include spatial translation of

the time sequences such that the similarity of two trajectories will depend on the distance between trajectories. In our approach the similarity was computed separately between time sequence projections on x-axis and y-axis. The main advantage of computing it in this way is a decrease in computation time from a cubic value to a quadratic one.

There are several parameters that need to be tuned for this computation and their values are application dependent. These parameters are: minimum ratio between the length of the shorter sequence and length of longer sequence, size of temporal matching interval and size of spatial matching interval.

Finally, the last step consists in applying a hierarchical clustering algorithm on the set of similarities computed before. In computing the distances between clusters, we tested three types of distances: minimum distance (Eq. 4.8), maximum distance (Eq. 4.9) and average distance (Eq. 4.10). The single-linkage algorithm failed to generate correct clusters (the clusters that were generated did not obey any of the requirements presented in the fourth paragraph of Section 5.2). The reason for this error was the fact that the distance defined, using the above-described similarity was not a metric and distance definition for inter-cluster distance is not suited for this kind of distances.

The other two algorithms generated pretty close solutions, even identical, depending how we pick the inter-cluster distance. Also, the generated clusters obey the initial restrictions: distance between trajectories whose ratio is smaller than the minimum ratio was set at a predefined big value. No two trajectories for which the distance was set at this high level were grouped together if the inter-cluster distance was small enough. For example, in Fig 5.13 – 5.15 can be seen that if inter-cluster distance is greater than 100, the clusters will stay unchanged until the magnitude of the distance will reach the level of 10^{11}. Another factor that will influence the structure of the clusters will be the size of the spatial matching interval.

An important fact is that even that we don't know at the beginning the final number of clusters, because of the manner we defined the distance between two trajectories, we know at least the minimum number of clusters. By changing the value of minimum ratio we also can change the minimum number of clusters.

Computing the distances between projections of trajectories might have a drawback also. Depending on the value of ε we can get a high similarity in one dimension and a low similarity for the other one. An example is presented in figure 6.1.

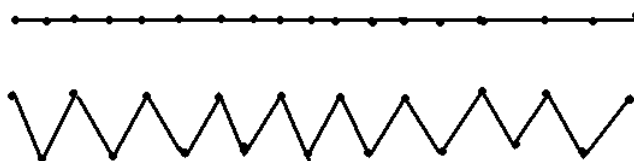


Fig 6.1

These kinds of cases, if there is an error, can be solved be decreasing value of ε. Nonetheless, there are cases when a trajectory does have a jagged appearance because of segmentation errors (e.g. when colors on the moving object and background are close, the size of the blob will change quickly from one frame to another and recorded trajectory will change accordingly; see Fig 5.1).

Appendix A

Proofs

Lemma 4.1: Given two trajectories *A* and *B*, with lengths *n* and *m* respectively, we can find the $LCSS_{2D}(\delta, \varepsilon, \rho, A, B)$ in $O(\delta\cdot(n+m))$ time.

Proof: The classic LCSS algorithm as described in [30] computes the longest common subsequence between two trajectories in $O(m\cdot n)$ time. The difference between the classic algorithm and our version consists in the matching process. While the LCSS algorithm in [30] allows matching between the elements of trajectories regardless of the distance between them, our version restricts the distance size to δ.

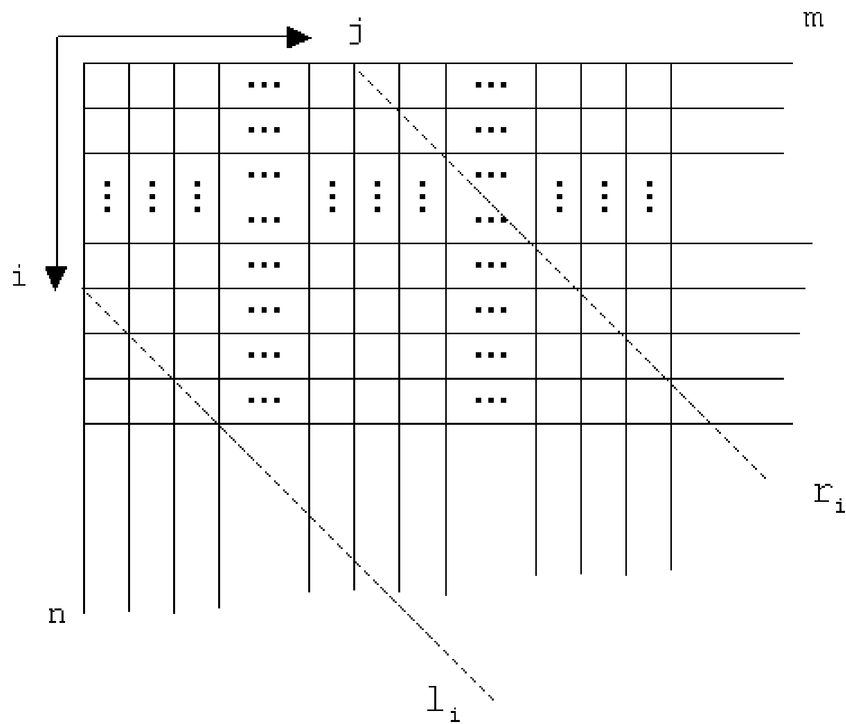


Figure A.1 Matching Table, Computed Using the $LCSS_{2D}(\delta, \varepsilon, \rho, A, B)$ Algorithm

Formula that we use for computing the matrix in Figure A.1 is:

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } |a_i - b_j| < \varepsilon \text{ and } |i - j| < \delta \\ \max(C[i, j-1], C[i-1, j]) & \text{otherwise} \end{cases} \quad \text{A.1}$$

where a_i and b_j are elements of the time sequences A and B.

In this proof we will show that our version of the LCSS algorithm doesn't need the values that in the Figure A.1 are on the left of line l_i or on the right of line r_i. How are l_i and r_i defined? Line l_i is the line that connects the $C[i, j]$ elements for which $|i - j| = \delta$ and $i > j$ while line r_i is defined as the line that connects the $C[i, j]$ elements for which $|i - j| = \delta$ and $j > i$.

For computing the matrix elements that are on the left of line l_i or on the right of r_i we will use either the first row or the third one of the Eq. (A.1). This means that their value is given either by the left or by the upper neighbor, but never by the upper-left neighbor. Elements that we need to analyze to prove our claim are the elements that are on the border of the region defined by the lines l_i and r_i.

	j – 1	j	j + 1
i - 1	C[i - 1, j - 1]	C[i - 1, j]	
i	C[i, j - 1]	C[i, j]	
i +1			C[i + 1, j + 1]

Lets consider the case when element $C[i, j]$ is on the l_i line. That means $i - j = \delta$.

a) if $|a_i - b_j| < \varepsilon$ then a_i matches b_j hence $C[i, j] = C[i - 1, j - 1] + 1$. But $C[i - 1, j - 1]$ is in the region between l_i and r_i.
b) if $|a_i - b_j| \geq \varepsilon$ then we will have to pick the maximum between $C[i, j - 1]$ and $C[i - 1, j]$. If $C[i - 1, j] \geq C[i, j - 1]$ we will pick $C[i - 1, j]$ which is still inside the desired region.
c) if $|a_i - b_j| \geq \varepsilon$ and $C[i - 1, j] < C[i, j - 1]$. This case is not possible. On the left of line l_i the value of an element $C[p, q]$ equals the value of the element "above" it, $C[p - 1, q]$. This means that $C[i, j - 1] = C[i - 1, j - 1]$ and consequently $C[i - 1, j - 1] > C[i - 1, j]$. But this is impossible because the way elements of our matrix are computed (elements on the rows and on the columns are monotonically increasing).

From a), b) and c) result that when we compute the value of an element on l_i border we use either elements inside the desired region or elements that are on the border. A similar reasoning can be made for the matrix elements that are on r_i border.

In conclusion, on row i of the matrix we need to compute the elements from $\max(0, i - \delta)$ to $\min(i + \delta, m)$ i.e. a maximum of $2 \cdot \delta$ elements. So the time needed to compute all the elements inside the desired region will be $O(\delta \cdot (n+m))$.

Bibliography

[01] Chris Stauffer, W.E.L. Grimson. “Adaptive Background Mixture Models for Real-Time Tracking” In *Computer Vision and Pattern Recognition*, pages 23-25, Fort Collins, Colorado, USA, June 1999

[02] Koichi Sato, J.K. Aggarwal. “Tracking Persons and Vehicles in Outdoor Image Sequences Using Temporal Spatio-Velocity Transform” In *Proceedings 2nd IEEE International Workshop on Performance Evaluation on Tracking and Surveillance,* pages xx-yy, Kauai, Hawaii, USA, December 9 2001

[03] Quming Zhou, J.K. Aggarwal. Tracking and Classifying Moving Objects from Video” In *Proceedings 2nd IEEE Interantional Workshop on Performance Evaluation on Tracking and Surveillance,* pages xx-yy, Kauai, Hawaii, USA, December 9 2001

[04] N.T. Siebel, S.J. Maybank. “Real-Time Tracking of Pedestrians and Vehicles” In *Proceedings 2nd IEEE International Workshop on Performance Evaluation on Tracking and Surveillance,* pages xx-yy, Kauai, Hawaii, USA, December 9 2001

[05] N.T.Siebel, S.J. Maybank. “The Application of Color Filtering to Real-Time Person Tracking” In *Proceedings of the 2nd European Workshop on Advanced Video-Based Surveillance Systems*, pages 227-304, Kingston upon Thames, United Kingdom, September 2001

[06] Tim Ellis, Min Xu. “Object Detection and Tracking in an Open and Dynamic World” In *Proceedings 2nd IEEE Internationl Workshop on Performance Evaluation on Tracking and Surveillance*, pages xx-yy, Kauai, Hawaii, USA, December 9 2001

[07] Ismail Haritaoglu, David Harwood, Larry S. David. “W^4: Who? When? Where? What? A Real Time System for Detecting and Tracking People” In *Proceedings of the Third Face and Gesture Recognition Conference*, pages 222-227, April 14-16, 1998, Nara, Japan

[08] Rómer Rosales, Stan Sclaroff. “Trajectory Guided Tracking and Recognition of Actions” Technical Report TR BU-CS- 1999-002, Boston University, September, 1999

[09] Christopher Wren, Ali Azarbayejani, Trevor Darrell, Alex Pentland. “Pfinder: Real – Time Tracking of the Human Body” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 780-785, July 1997, vol 19, no 7

[10] A. Azarbayejani, A. Pentland. “Recursive Estimation of Motion, Structure and Focal Length” in *Pattern Analysis and Machine Intelligence,* pages 562-575, June 1995, vol 17, no 6

[11] Tony Jebara, Ali Azarbayejani, Alex Pentland. “3D Structure from 2D Motion” in *IEEE Signal Processing Magazine*, pages xx-yy, May 1999, Vol 16, no 3

[12] Markus Koler. “Using the Kalman Filter to track Human Interactive Motion – Modeling and Initialization of the Kalman Filter for Translational Motion – Research Report No. 629/Feb1997, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Germany

[13] Richard Hartley, Andrew Zisserman. “Multiple View Geometry in Computer Vision” Cambridge University Press, September 2000

[14] Richard Duda, Peter E. Hart, David G. Stork. “Pattern Classification” Wiley-Interscience, 2001

[15] Ramesh Jain, Rangachar Kasturi, Brian G. Schunck. “Machine Vision” McGraw-Hill, Inc. 1995

[16] Michalis Vlachos, George Kollios, Dimitrios Gunopulos. “Discovering Similar Multidimensional Trajectories” In *International Conference on Data Engineering 2002*, pages 673-684, San Jose, California, 2002

[17] Béla Bollobás, Gautam Das, Dimitrios Gunopulos, Heikki Mannila. “Time-Series Similarity Problems and Well-Separated Geometric Sets” in *Proceedings of the 13th Symposium on Computational Geometry*, pages 454-456, Nice, France, 1997

[18] Gautam Das, Dimitrios Gunopulos, Heikki Mannila. “Finding Similar Time Series” in *Proceedings of the First Principles of Data Mining and Knowledge Discovery Symposium*, pages 88-100, Trondheim, Norway, 1997

[19] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, Kyuseok Shim. “Fast Similarity Search in the Presence of Noise, Scaling and Translation in Time-Series Databases” in *Proceedings of Very Large Data Bases,* pages 490-501, Zürich, Switzerland, September, 1995

[20] Tolga Bozkaya, Nasser Yazdani, Meral Özsoyoğlu. “Matching and Indexing Sequences of Different Lengths” in *Proceedings of the Conference on Information and Knowledge Management*, pages 128-135, Las Vegas, 1997

[21] David A White, Ramesh Jain. “Algorithms and Strategies for Similarity Retrieval” Technical Report VCL-96-101, Visual Computing Laboratory, Univ. of California, San Diego, La Jolla, CA, July 1996.

[22] Christos Faloutsos, King-Ip (David) Lin. “*FastMap*: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 163-174, Vol.24, no.2, June, 1995

[23] Christos Faloutsos, M. Ranganathan, Yannis Manolopoulos. “Fast Subsequence Matching in Time-Series Databases” in *Proceedings 1994 ACM SIGMOD Conference*, pages 419-429, Minneapolis, MN, 1994

[24] Rakesh Agrawal, Christos Faloutsos, Arun Swami. “Efficient Similarity Search in Sequence Databases” in *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms*, pages 69-84, Chicago, Illinois, 1993

[25] Nasser Yazdani, Meral Özsoyoğlu. “Sequence Matching of Images” in *Proceedings of the 8th International Conference on Statistical and Scientific Database*, pages 53-63, Stockholm, Sweden, June, 1996

[26] Eamonn Keogh. "Exact Indexing of Dynamic Time Warping", in *Proceedings of the 28th Very Large Data Bases Conference*, pages 406-417, Hong Kong, China, 2002
[27] Sang-Wook Kim, Sanghyun Park, Wesley W. Chu. "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases" in *Proceedings of the 17th International Conference on Data Engineering*, pages 607-614, Heidelberg, Germany, April, 2001
[28] Byoung-Kee Yi, H.V. Jagadish, Christos Faloutsos. "Efficient Retrieval of Similar Time Sequences Under Time Warping" in *Proceedings of the 14th International Conference on Data Engineering*, pages 201-208, Orlando, FL, February, 1998
[29] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, Sharad Mehrotra. "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases" in *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 151-162, Santa Barbara, CA, May, 2001
[30] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. "Introduction to Algorithms", The MIT Press, 1990
[31] Robert Grover Brown, Patrick Y.C. Hwang. "Introduction to Random Signals and Applied Kalman Filtering", John Wiley & Sons, Inc. 1997
[32] Anil K. Jain, Richard C. Dubes. "Algorithms for Clustering Data", Prentice-Hall, Inc. 1988
[33] Berthold Klaus Paul Horn. "Robot Vision", The MIT Press, 1997
[34] Michael Isard, Andrew Blake. "CONDENSATION – conditional density propagation for visual tracking", in *International Journal of Computer Vision*, pages 5-28, 29(1), 1998
[35] Performance Evaluation of Tracking and Surveillance Workshop, PETS 2001, http://pets2001.visualsurveillance.org