Video-Based Human Body Action Recognition for Games

Master's Theses in Telematics

at

Graz University of Technology

Telematics Master Studies[1]

submitted by

Markus Murschitz[2]

Institute for Computer Graphics and Vision (ICG),
Graz University of Technology
A-8010 Graz, Austria

25th August 2011

Advisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Co-Advisor: Dipl.-Ing. Thomas Mauthner

[1] Course ID: F-066-411
[2] Student ID: 0130596

Ganzkörper-Gesten basierte Steuerung eines Komputerspiels

Masterarbeit in Telematik

an der

Technische Universität Graz

Masterstudium der Telematik[3]

vorgelegt von

Markus Murschitz[4]

Institut für Maschinelles Sehen und Darstellen (ICG),
Technische Universität Graz
A-8010 Graz

25. August 2011

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter:	Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Mitbetreuender Assistent:	Dipl.-Ing. Thomas Mauthner

[3]Course ID: F-066-411
[4]Matrikelnummer: 0130596

Abstract

If a person performs an action (like walking), and is captured by a camera, it turns out that the captured video contains patterns corresponding to the action performed. Such is exploited by the field of Human Body Action Recognition, where videos are classified using computer vision algorithms. Human Body Action Recognition is an active research topic. As with various modern-day human-computer-interface technologies one of the first applications is the gaming industry.

The aim of this work is to utilize such classifications to control a computer game, where a specific action corresponds to a specific keystroke pressed in the game. While many published works address the topic of video-based action recognition, there are not many works, which are able to perform it in real time, which is a crucial requirement for any gaming application. Another requirement is to be able to the detect the number of repetitions an action has been performed.

In this work this two requirements are addressed. The real-time requirement is addressed by exploiting the massively parallel computing capabilities of modern graphics cards for dense feature extraction and actor detection. Where the features are Histograms of Oriented Gradients (HOG) and Local Binary Patterns (LBP) on appearance and the Histogram of Flow-Orientations (HOF) and Local Binary Patterns on Flow-magnitude (LBFP). The motion and flow information of the actor are transformed into a prototype per frame. The sequence of prototypes is analyzed by subsequence-matching to known action-specific prototype sequences. This results in an action classification and information about their temporal alignment, which is used to perform repetition detection. The repetition detection and action classification are performed by utilizing Dynamic Time Warping (DTW) as a distance measure.

Evaluations are performed on several public available datasets and compared to results of other works. It turns out that the system leads to comparable (but slightly inferior) results which are be accomplished in real time. Finally a prove of concept is given by incorporating the full evaluation pipeline with a game, where the evaluation pipeline works as a substitute for a keyboard.

Keywords:

Computer Vision, Human Body Action Recognition, Human Computer Interaction, Histogram of Oriented Gradients, Local Binary Patterns, Vision Controlled Games, Dynamic Time Warping

Kurzfassung

Führt ein Mensch eine Aktion (wie beispielsweise Gehen) aus und wird dabei von einer Kamera aufgezeichnet, so lässt sich in dem entstandenen Video eine Struktur erfassen, die spezifisch für diese Aktion ist. Mustererkennung kann diese Struktur erkennen und daher Videos als bestimmte Aktionen klassifizieren.

Das Ziel dieser Arbeit ist es, solche Klassifikationen zur Steuerung eines Computerspiels zu nutzen, wobei eine Aktion einem Tastendruck gleichgesetzt werden soll. Aus dieser Anwendung ergeben sich zwei Vorgaben für das System: Einerseits muss es echtzeitfähig sein und andererseits muss es Wiederholungen von Aktionen erkennen können. Es existiert bereits eine breite Palette an Publikationen zum Thema Mustererkennung von menschlichen Ganzkörper-Gesten und Aktionen, aber nur wenige dieser Arbeiten klassifizieren in Echtzeit.

In dieser Arbeit werden existierende Systeme analysiert, und ein System präsentiert, dass diese Aufgabe erfüllt. Es erlangt Echtzeitfähigkeit dadurch, dass die nötigen Features und die Detektion der Person auf der Grafikkarte umgesetzt werden. Die Features sind Histograms of Oriented Gradients (HOG), Local Binary Patterns (LBP), Local Histograms of Oriented Flow-Magnitudes (HOF) und Local Binary Flow Patterns (LBFP). Weiters wird für jeden Video-Frame ein Prototyp generiert, der sowohl die Bewegungsinformation als auch das Erscheinungsbild der Person codiert. Die für ein Live-Video entstehende Prototypenkette wird mit bekannten aktionsspezifischen Prototypen-Sequenzen verglichen. Dies wird mittels Dynamic Time Warping (DTW) realisiert. Es werden Subsequenzen in der Kette detektiert und klassifiziert. Das Resultat sind die Klassifikation und deren zeitliche Ausdehnung. Diese Information wird genutzt, um vollständig ausgeführte Aktionen und damit auch deren Wiederholungen zu erkennen.

Das System wird mittels existierender Datensätze evaluiert, und die Resultate werden mit denen von anderen Arbeiten verglichen. Es zeigt sich, dass das trotz Echtzeit-Fähigkeit vergleichbare (wenn auch nicht bessere) Resultate erzielt werden können. Außerdem wird gezeigt, wie das System ein Computerspiel steuert.

Schlagwörter:

Maschinelles Sehen, Ganzkörper-Gesten Erkennung, Mensch-Computer-Interaktion, Histogram of Oriented Gradients, Local Binary Patterns, Vision Controlled Games, Dynamic Time Warping

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

Abbreviations and Symbols

∇	Gradient
μ	Mean value of a number of samples for k-means
$\mathbf{A}$	Orientation of a SVM decision surface
$\mathbf{D}$	The Dynamic Time Warping matrix
$\mathbf{f}$	Feature
$\mathbf{H}$	Matrix of coefficients (factorization)
$\mathbf{I}$	Identity matrix
$\mathbf{P}$	Projection matrix for changing a base (factorization)
$\mathbf{V}$	Original data for a factorization
$\mathbf{W}$	Warping Path (dynamic time warping) Matrix containing base vectors (factorization)
$\mathbf{w}$	An element consisting of a pair of indexes for a DTW-path
$\mathbf{x}$	A sample to be classified, or a position in the image
$\mathbf{X}[t]$	Image as signal of timestep t
b	Offset of a SVM decision surface
$dist(.,.)$	Any kind of distance between two data points
h	Image height
p_x	Patch width
p_y	Patch height
t	Usually time, target vector for Support Vector Machine
w	Image width
$y(\mathbf{x})$	Linear classification model for a sample $\mathbf{x}$

AL-DTW	Adaptive Length Dynamic Time Warping
API	Application Programming Interface
CLBP	Circular Local Binary Pattern
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DTW	Dynamic Time Warping
EM	Expectation Maximization
GP-GPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HOF	Histogram of Oriented Flow
HOG	Histogram of Oriented Gradients
ICG	Institute for Computer Graphics and Machine Vision
LBFP	Local Binary Patterns of Flow Magnitude
LBP	Local Binary Pattern
LDA	Linear Discriminant Analysis
LSB	Least Significant Bit
LSVM	Linear Support Vector Machine
MEX	MATLAB Executable
MSB	Most Significant Bit
NMF	Non Negative Matrix Factorization
PCA	Principle Component Analysis
ROR	Rotate Right (bitwise rotate right operator)
SIFT	Scale Invariant Feature Transform
SVM	Support Vector Machine
TV-L1-Flow	Total Variation L1 Flow

Acknowledgements

I am indebted to my advisors Thomas Mauthner and Horst Bischof as well as to all the other members of the Institute of Computer Graphics and Vision who have provided me with a lot of expertise, help, feedback and motivation during the course of my work. Especially I want to thank Jakob Santner for his indispensable help with the treat of programming the graphics processor and Peter Roth for his generous help and feedback.

Finally I would like to thank my family and my dear girlfriend Lisa who supported me in my whole studies and especially during the time of my thesis.

Markus Murschitz

Graz, Austria, 2011-08-25

Chapter 1

Introduction

Since the early days of photography people showed interest in how bodies move. They built various machines to display and analyze the motion of humans and animals. Figure 1.1 shows one of these studies. One can recognize a certain order in such studies, which is not only a work of art as well as a scientific analysis.

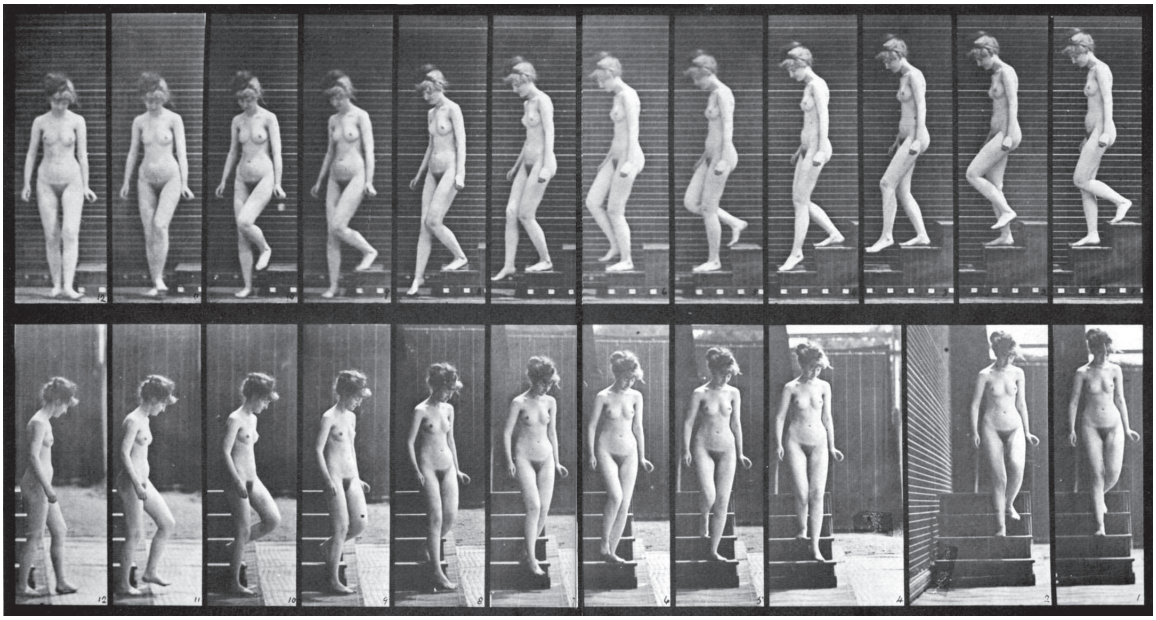


Figure 1.1: Woman Walking Downstairs from Eadweard Muybridge [Muybridge, 1955]

The idea of Human Body Action Recognition is to exploit this order, to employ the re-occurrence of certain patterns in human motion while performing an action. The body goes through a signature sequence of poses, which is independent of the individual but corresponds to the kind of action performed.
Once such an action can be classified using vision technology, applications are various, see [Turaga et al., 2008].
Such are:

- **Behavioural biometrics** deals with the unique identification of a human based on its behavioral

cues, which allows an identification without direct interaction.

- **Content-based video analysis** wants to overcome the problem of increasing amounts of video information in our everyday live by developing efficient indexing and storage technologies.
- **Surveillance** raises the same problem. The security personal watching live video surveillance are over-strained by the vast amount of data to process. Therefore systems have to be created which perform a preselection of interesting scenes. Even a complete substitution for a human operator might be possible.
- **Animation and synthesis** of human body movements are a topic of increasing interest in entertainment industries as gaming and animation.
- Last but not least, there is **human-computer-interaction** where actions are used to control a computer in an non-verbal way. As the body movement is the most important mode of non-verbal communication a more natural way of communicating with a computer can be accomplished. This is the target application of this work.

1.1 Scope of this work

The scope of this work is to present and evaluate a system allowing human computer interaction by gestures. The final goal is to use a single camera connected to a standard personal computer to recognize and categorize actions. Once this actions are classified a specified signal for each action is send to the a gaming application which reacts dependent on the signal.

Additionally the end of an action is detected to be able to generate a single keystroke-like trigger per each repetition of an action. Hence in a computer-game a user is accustomed to a certain behavior: Multiple keyboard-hits are expected to have the effect of the according command to be executed multiple times. Therefore, if an action recognition system is intended

to substitute a keyboard to a certain amount, it has to be able to detect repetitive input.

The novelties are, firstly, that the computationally intense features necessary to detect and classify the actions are computed on the graphics card to gain real time. And secondly, to perform the repetition detection. Another detail is that the action recognition is split in two steps, a actor-detection and an action-classification step, both working with the same features. In the detection step the interesting image-region is found while in the classification step extracted information of multiple frames is combined to classify the action.

1.2 Action Recognition Problems and Terminology

As already explained in an intuitive way the task of action detection is a problem of pattern recognition. [Bishop, 2007] describes the field of pattern recognition as follows:

> The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take

actions such as classifying the data into different categories.

The theory distinguishes between classification and regression. The following example application will focus on this difference.

Classification

A simple sample task will serve to explain the principles behind classification. Assume we want to distinct between three cases of the position of a human body by looking at a single photography of the human.

The positions are an upright standing person facing us, a sidewards bending person, and a person which is running side-wise. We can encode this entities with numbers by enumeration as shown in figure 1.2. A set of labels contained in the dataset is denoted as C and corresponds to a set of classes which was trained on (in this example $C = \{1, 2, 3\}$).

Figure 1.2: A body position recognition example - showing samples of the three classes to distinct. The classes are labeled with the encircled numbers. (generated from Weizmann action dataset [Blank et al., 2005])

Assume a collection of pictures $\{\mathbf{X}_1, ..., \mathbf{X}_N\}$ with all of these poses performed by different people where for each picture the label $\{c_1, ..., c_N\} \in C$ is known. This collection will further on be referred to as training dataset.

The result of running a pattern recognition algorithm can be defined as a function $f_{class}(\mathbf{X})$ which transforms an image $\mathbf{X}$ into a label and therefore a class prediction.

The quality of the trained function is evaluated by using another dataset, the test set. The labels of this set are not known to the system. If the system can produce the right labels by applying the former trained function the system is called to be generalizing. This is the objective in pattern recognition [Bishop, 2007].

Regression

Regression on the opposite is not only interested in a label but also in a value representing a probability measure of a sample to be of a certain class. Therefore the output values of $f_{regr}(\mathbf{X}))$are not discrete labels representing an entity but one probability value $p(\mathbf{X}|c)$ for each class, where c denotes each class

contained in C and $\sum_{c \in C} p(c|\mathbf{X}) = 1$. Such can be converted in a probability measure $p(c|\mathbf{X})$ by the Bayes-theorem $p(c|\mathbf{X}) = p(\mathbf{X}|c) \cdot p(c)/p(\mathbf{X})$.

Incorporating temporal Information

The simple example given above is working on single images and not a stream of images, a video. In action recognition the temporal occurrence of patterns is an inherent part of the problem and has to be incorporated in the recognition process itself.

To follow up with the already introduced notation: In general the problem of a sequence classification or regression can be represented as a function $f(\mathbf{X}[t])$ where $t = 1...N_{video}$ and N_{video} is the length of the video[1].

If we are talking about a live system (where the frames are captured by a camera one at a time) a temporal window approach is more appropriate. Where the pattern recognition function uses a certain amount of frames N_{win} in the past until now to compute a prediction for the current frame. Therefore $f(\mathbf{X}[t-i])$ where $i = \{0, ..., N_{win}\}$ represents the classification process performed at time t.

Actions Activities and Gestures

In existing literature the terms action and activity are used interchangeable. Some authors like [Turaga et al., 2008] are distinguishing them by their complexity. Where the term action is used to refer to a simple motion pattern like walking or swimming usually performed by a single person whereas activities are sequences of actions usually performed by multiple individuals. They are usually of longer duration and more complexity like shaking hands or a football team scoring a goal. The boundary between these two is a soft one.

Other authors like [Nater et al., 2010] refer to Turaga's actions as micro-actions and Turaga's activities are called actions. In this work the first naming convention of Turaga will be used.

Additionally the term gesture occurs. The difference is that a gesture either is a sort of non verbal communication, or a by-product of a spoken message whereas an activity like walking refers to an action performed for its output not its semantic. In terms of computer vision and machine learning this distinction is irrelevant since it is only a question of intention and similar methods are applied in both cases. [Laurel, 1990] emphasizes on the difference between a gesture and a keystroke as follows:

> A gesture is a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger in its way to hitting a key is neither observed nor significant. All that matters is which key was pressed.

1.3 Structure of this work

Since Video-based Human Body Action Recognition finds its way into our living rooms nowadays, there are plenty of publications dealing with this topic. Therefore the following chapter (Chapter 2) is giving

[1] The convention of writing $\mathbf{X}[t]$ indicates that the image $\mathbf{X}$ is a signal in time whereas $\mathbf{X}(x, y) \equiv \mathbf{X}(\mathbf{x})$ with $\mathbf{x} = (x, y)^T$ refers to pixel access.

an overview of some state of the art technologies in the field. A typology of action recognition systems is presented. Additionally the main influences for this work are discussed.

Chapter 3 introduces some of the theory and mathematics which are used in the proposed system in detail. An experienced reader might skip various subsections of this chapter, but it is strongly recommended to read Section 3.7 since it presents specialized modifications to the Dynamic Time Warping Algorithm.

The second part (Chapter 4) of this work aims to describe the implemented system where Section 4.1 gives a detailed systematic overview. Implementation details are presented and modifications to existing algorithms to be feasible for the given task are explained. This chapter presents the practical part of this work and focuses on the novelties it will refer to specific theoretical sections in Chapter 3 whenever needed.

Chapter 5 is all about evaluation and prove of concept. It presents the evaluation of the proposed algorithm on in the field known datasets and compares the results to the work of others. Speed and Performance evaluations are presented and the design of the live system and its connection to the game is discussed.

Chapter 6 shows how applications are controlled by the evaluation-pipeline and is therefore all about proof of concept.

Chapter 7 summarizes the learned lessons and proposes future work in the field.

A more detailed listing of references to specific topics can be found in the upcoming Section 1.4, which will summarize the taken approach for action-recognition.

1.4 A Strategy for Action Recognition

There are many possible ways to approach the problem of action recognition (see Section 1.5). One possible approach shall be introduced in this section, allowing the reader to get a general idea of the system, and to be able to refer to the rest of the work in a structured way. For a more detailed system description see Section 4.1. The necessary training for such a system is the topic of Section 4.2.

The action recognition method which is the subject of this work splits the classification task into four mayor tasks:

1. Feature extraction
2. Region of interest (ROI) detection
3. Prototype extraction
4. Prototype sequence matching

Each of the subtasks shall be explained in the following while their systematic interaction is visualized in Figure 1.3. Only the classification process is visualized once more in Figure 1.4 emphasizing the temporal aspect.

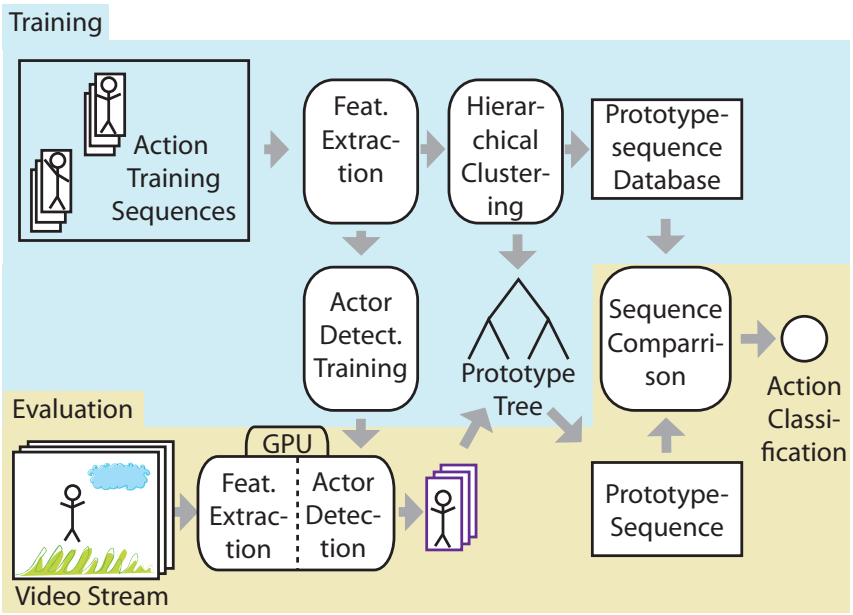


Figure 1.3: Overview of the action recognition strategy - During the training phase features of labeled action-videos are extracted. They are clustered hierarchically to generate a tree structure and are used to train an actor-detector. The leaves of the tree are the prototypes. A prototype sequence database records the occurring prototype-sequences and their label. During evaluation the same features are extracted for the video stream and are used for actor-detection. For each frame a prototype is extracted by tree traversal. The resulting prototype-stream is compared to the sequences in the database. Resulting in a classification result.

Feature Extraction

Until now $\mathbf{X}$ denoted an image meaning the values of all of its pixels. It turns out that certain types of preprocessing of an image to an intermediate result, further on called a feature $feat(\mathbf{X})$, are beneficial for performing classification.

There are two major types of features, dense features and sparse features. The distinction is, that a dense feature is a vector computed for each pixel of the input image $\mathbf{X}$, whereas a sparse feature is only calculated for so called points of interest. In this work the focus will be on dense features.

In both cases, dense and sparse features, the feature is usually calculated from a pixel value and local neighborhood of the pixel. There is a vast amount of features proposed in the field. Mostly a feature aims for a certain property. Such properties are rotation-invariance, scale-invariance, gray-scale-invariance and others. These properties also define the range of applications for which they are feasible. In our case the most important property is the gray-scale invariance. Gray-scale invariant features are features which are approximately equal for a scene captured at different lightning. Some features will be discussed in detail. Namely the Local Binary Patterns in Section 3.2, and the Histograms of Gradients in Section 3.3. Both of them will not only be calculated for the appearance (the image itself) but also for the movement between two consecutive images which is referred to as optical flow and explained in Section 3.1. The necessary adaptations to the features are discussed in Section 3.3.1 and Section 3.2.1

One of the novelties in this work is to calculate such features near real time. Therefore they are computed by the graphics card in a parallelized algorithm resulting in a significant speedup. The implementation

details can be found in Section 4.4 and 4.5. While general considerations for calculations on the graphics card are discussed in Section 4.3. Such features are in their nature of high dimension. A reduction in dimensionality is performed by the Non Negative Matrix Factorization (NFM). Such dimensionality reduction is the topic of Section 3.4 where NMF is compared to the Principle Component Analysis (PCA). The according implementation details are subject to Section 4.3.4.

Region of Interest or Actor Detection

The amount of information within an image is vast. To make sense of it in terms of the proposed classification system only a part of it contains the information to process. This part is called the region of interest (ROI). It can be represented as a rectangular region called *bounding box*. The specific task of actor detection is fulfilled if such bounding box encloses the actor inside the image. In our case this is performed by a linear Support Vector Machine (LSVM) the according theory will be the topic of Section 3.5, while implementation details can be found in Section 4.6.

Prototype Extraction

Generally speaking a prototype in terms of computer vision is a representative sample of a set of values. To extract a prototype for a given region of interest one has to find a representative, out of a set of previously determined representatives, which is most similar to the features calculated from the ROI. In this case the prototypes are cluster centers which are generated by employing the k-means algorithm on the features during the training phase (explained in detail in Section 3.6). A prototype can be represented by an unique identifier, which means to reduce the information inside the ROI to a single value per frame. Implementation details can be found in Section 4.7.

Prototype Sequence Matching

If such prototypes are concatenated over time - one prototype per frame - a sequence of prototypes represents the content of the video. Such sequences of prototypes have also been created during the training-phase. In the evaluation-phase the most similar sequence has to be found. To determine similarity a specialized distance measure is employed. In this case the Dynamic Time Warping (DTW) Algorithm is used because it is able to reasonably compare sequences of different length even if they are ”warped” in time. Such will be explained in detail in Section 3.7. A modification of the algorithm the *Adaptive Length Dynamic Time Warping* (AL-DTW) is presented in Section 4.8.1. The label according to the sequence with the minimum distance (in terms of AL-DTW) is the classification result. The modified algorithm gains minimal distances over time if the end of an action is reached. This fact is exploited to detect the end of an action. Therefore the number of repetitions of an action can be determined.

Due to problems for real life applications another method for sequence comparison is implemented: The SPRING [Sakurai et al., 2007] algorithm. It is a modified form of DTW-comparison specialized for subsequence matching on streaming data which is explained in detail in Section 4.8.2. Sections 4.8.1 and Section 4.8.2 are deliberately left out of the theoretical part, since both algorithms are motivated by implementation details (to be fast) rather than theoretical concepts.

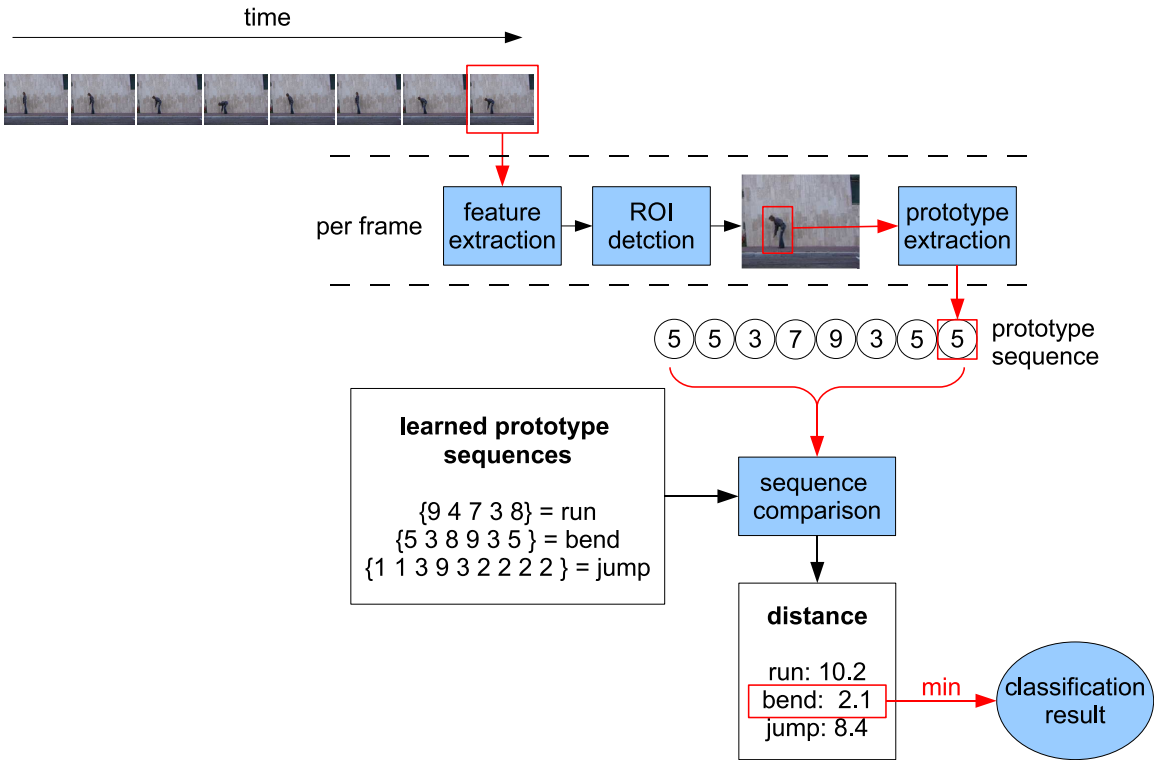


Figure 1.4: A sample action classification for streaming data - The graphic shows a system which performs the classification of three actions "run", "bend" and "jump" during classification. For each frame of a video features are extracted. The result is used to detect the region of interest (ROI) in the image - the actor (indicated by the red bounding box). A prototype is extracted for this region. It is represented by a unique prototype identifier (a number). Over time such prototypes form a sequence. This sequence is compared to previously (during the learning phase) recorded prototype sequences. For each sequence of the database a distance to the actual prototype sequence is computed. The sequence label according to the minimal distance (in this case "bend") is the final classification result.

1.5 Structure of Action and Activity Recognition Systems

When analyzing existing action and activity recognition systems a certain three level structure can be observed. Each level is dependent on the previous one [Turaga et al., 2008]:

1. **Low-level operations** are low-level image / video data driven modules, like background-foreground-segmentation, feature extraction, object detection and simple forms of tracking (bottom up).

2. **Mid-level operations** are the action recognition modules. Usually they have been trained[2] and therefore incorporate some model driven information (top down).

3. **High-level operations** are reasoning engines which encode activity semantics based on the former detected actions (top down).

[2]There is the exception of unsupervised action analysis where no labels are given to the samples at all, but these are not the focus of this work.

The focus of this work will be on the low-level and mid-level. If we look at the already introduced strategy for action recognition we can assign the feature extraction and ROI detection to the low-level operations, while the prototype extraction and prototype sequence matching are high-level operations.

Chapter 2

Related Work

In this chapter the related work is listed. Therefore this section will list the major influences for the presented method of action-recognition and the relation to other existing work. Then the influences for the real-time implementation specific resources will be listed separately, before discussing the typology of the field of action-recognition. In this part also works are discussed, that can not be seen as directly influencing this work, but being part of the history of action-recognition systems in general.

Major Influences

Following the literature over the last years, activity recognition can often be seen as classification of videos, constrained to contain only one significant action. In fact, many common datasets [Gorelick et al., 2005, Schuldt et al., 2004, Lin et al., 2009, Rodriguez et al., 2008] are fragmented in that way, and often classification results per video are given. More realistic data is given with the dataset of [Ke et al., 2007]. Beside the fact that detection could be an elaborate problem, the realistic scenario of consecutive variations of actions is unattended . Furthermore, applying action recognition for human computer interaction requires the recognition of repetitions as mentioned in the previous section. Therefore we focus on approaches classifying activities based on sequences of action primitives or "action words", shown by [Weinland and Boyer, 2008, Thurau and Hlavac, 2008, Lin et al., 2009, Yao and Zhu, 2009], and other real-time systems [Reddy et al., 2009, Yu et al., 2010].

Weiland and Boyer [Weinland and Boyer, 2008] used foreground segmentations to create a set of silhouette exemplars, so called key-poses, using a forward selection process. The final action description is achieved by comparing the occurrence frequency of key-poses in a video. Although they presented excellent results, they completely neglected the temporal ordering of prototypes within a sequence and showed only recognition results on complete videos. To incorporate temporal context in a prototype-based representation, Thurau and Hlavac [Thurau and Hlavac, 2008] introduced n-grams models. They define sub-sequences of n frames and describe the transitions between prototypes by n-dimensional histograms. However, the required number of samples to fill the n-dimensional histograms is high and the temporal ordering is very strict. Furthermore, the representation of n-grams is getting difficult if $n > 3$. Experimentally, they showed state-of-the-art results on sequences with lengths of around 30 frames.

Since shape information clearly gives only limited information on single-frame basis [Lin et al., 2009]

proposed to create prototypes in a joined shape-motion space using binary foreground silhouettes for shape and flow features as introduced in [Efros et al., 2003]. They employ k-means to cluster the combined shape-motion-features. Then they build a tree structure upon these clusters. During evaluation this tree is traversed by depth first search. The temporal information is incorporated using Dynamic Time Warping (DTW). Although DTW is a powerful method for aligning temporal sequences, as a drawback it only compares one sequence to another and cannot handle transitions between different actions. Again only results on sequence level are shown. Continuous videos with varying activities have not been shown in their work. The problem of aligning training sequences to longer test sequences is handled by removing redundant segments at the start and end of the DTW path. They also propose an efficient way for calculating the DTW-costs from prototype sequences. Which is performed by calculating all prototype to prototype distances during training and using such as a lookup-table during evaluation. By doing so they present an efficient method for generating prototypes out of features and calculating DTW-costs on such sequences. Nevertheless they do not evaluate the overall evaluation pipeline for speed.

[Yao and Zhu, 2009] presented an approach to solve the actor-detection and recognition part in one dynamic space time warping approach. Although their main goal was towards weakly supervised learning of actions, they pointed out the problem of temporal alignment and assumed manually temporal aligned actions during training. During testing the alignment problem is bypassed by working on temporal overlapping subsequences with around 2-3 minutes computational time per subsequence.

An computational efficient alternative to patch-based approaches is the usage of spatio-temporal-interest-points (STIP) for representing activities. Reddy *et al*. [Reddy et al., 2009] used an efficient hierarchical classification framework and accumulated classification votes of detected key-points for activity recognition. Their system has real-time capabilities but of course the limitation that no misleading motions have to appear in the background. Applying very efficient key-point extraction and classification methods Yu *et al*. [Yu et al., 2010] proposed an approach also capable to handle activities with more complex spatio-temporal relations. Although they have a very short response time, they tested on subsequences with a length of around two seconds, which is too much for real-time user feedback.

Zhou [Zhou and Torre, 2009] employ a combination of Canonical Correlation Analysis and DTW called Canonical Time Warping (CTW) to find temporal alignments of facial expressions, human body motion capture data and synthetic data. CTW is a feature selection mechanism which also enables a comparison of features with a different number of dimensions. They show that CTW is outperforming the DTW in terms of temporal alignment of sequences. They also perform local time-warping by a localized version of CTW allowing multiple local spacial deformations. In a previous work Zhou also presents a method called *Aligned Cluster Analysis*(ACA) (in [Zhou et al., 2008]) which performs temporal segmentations of human motion. Meaning a video composed of different actions performed, is analyzed to find segments corresponding to actions. Such is done ofline, meaning the complete video has to be captured already. They also have to decide for the number of action-segments they want to find in advance. ACA is an extension of kernel k-means. It combines clustering with dynamic time alignment performed by dynamic programming. They show promising results, for motion capture data and synthetic data, but admit the necessity of further improvement.

[Roth et al., 2009] use Histograms of Oriented Gradients and Histogram of flow Orientation and a Non Negative Matrix Factorization to generate features which are classified by multiple *Cascaded Linear*

Discriminant Analysis classifiers. NMF turned out to enhance the performance of the linear classifier for the purpose of action recognition. In a later work [Mauthner et al., 2011] they propose the additional usage of Local Binary Patterns on flow magnitude and appearance to create four different prototype trees. They discuss the fact that the significance of motion and appearance varies for different actions. And that it also varies during the performance of an action. Therefore they introduce a temporal weighting scheme for accumulating the information of these four prototypes over time, in respect to their significance. Leading to fast and accurate per frame results for action classification. But no temporal alignment of sequences is performed. The features used are the same as will be presented in this work. HOG was already used for human detections since Dalal *et.al.* presented their human detector in [Dalal and Triggs, 2005]. They are originated in the definitions of the SIFT descriptor, which was introduced by [Lowe, 2004]. Both, SIFT and HOG, are encoding edge orientations of small image regions in histograms which are normalized by the histogram energy of neighboring histograms. Therefore they present themselves as stable and discriminative features, even in cluttered backgrounds and under different illumination.

LBP are outstanding features describing texture information and where used for human detection for example by [Wang et al., 2009]. They are based on a texture model which represents texture as a histogram of binary codes. The binary codes are established by comparison of multiple grayscale neighboring values according to a specific pattern.

Also some of the publications from the field of unsupervised activity recognition are interesting for this work. [Zhong et al., 2004] use tree based prototypes to detect unusual activities in videos. They employ the co-occurrence of segments of videos which satisfy the transitive-closure constraint. [Nater et al., 2010] propose a data driven unsupervised method for unusual activity recognition which creates an appearance prototype tree and a sequence analyzing tree.

Implementation Specific Influences

[Müller, 2007] introduces the basics of dynamic time warping in a structured way that also shows the usage of DTW for subsequence-matching.

FastDTW [Chan, 2007] is an approximation of DTW that has linear time and space complexity. It follows a coarse to fine multi-level approach to compute the DTW-warping path. It performs fast and accurate for ofline problems with very long time-series but is not equipped to handle streaming data. It used by [Lin et al., 2009].

[Sakurai et al., 2007] introduce a method for subsequence matching based on dynamic time warping called SPRING. It is specialized on streaming data and has linear time and memory complexity per time step while reusing all the data which has been computed during the past time-step.

In terms of real-time feature calculation the main influences have been the *fastHOG* of [Prisacariu and Reid, 2009] and the OpenCV library [Bradski, 2000] which was extended in the Robot Operating System (ROS) version [Quigley et al., 2009] to be able to calculate HOG in real-time.

In terms of writing algorithms for the GPU the CUDA-Programming-Guide [NVIDIA, 2009b] contains guidelines for writing algorithms on GPUs with the NVIDIA specific CUDA-language. It presents performance optimization strategies. Which are how to maximize parallel execution to achieve maximum utilization, how to optimize memory usage to achieve maximum memory throughput and how to opti-

mize instruction usage to achieve maximum instruction throughput.

Typology of existing Action Recognition Systems

If one looks further in the past it turns out, that the palette of different approaches and techniques for action recognition are wide spread. Therefore a recapitulation shall be given. Not each work will be explained separately and in detail, but approaches will be grouped by their general idea, leading to a *Typology of Action Recognition Systems* which will still be referencing to specific publications.

[Turaga et al., 2008] divides the field of action recognition into three groups of classes which correspond to the former (in Section 1.5) defined three levels. As this work only deals with the first two levels (low and mid-level) only these typologies will be presented.

The following listing shows a grouping according to the low-level layer:

1. Point Trajectories
   Point trajectories describe the path of a single point or an object over time. Certain properties like speed and velocity are used for the following layers. Very accurate tracing algorithms are needed in order to compute unambiguous motion trajectories. Noise and occlusions tend to make this a hard task on its own. For a detailed survey see [Cedras and Shah, 1995].

2. Blobs and Shape
   It is common practice to perform background subtraction to end up in a binary image. The result is a silhouette figure. From this silhouette figure boundaries and skeletons can be extracted as well as simple blobs. These geometric entities are further on encoded in an appropriate way for the following layers. These approaches can only be as good as the background subtraction, therefore they are not robust against background noise.

3. Filter Responses
   Several other approaches exist using spatio-temporal features like the derivative of Gaussian filter in [Zhong et al., 2004] or a 3D version of the Harris corner detector in [Laptev, 2005] which is a sparse representation. Such methods work on the video as 3D-volume in contrast to working on a sequence of separately analyzed images. A similar approach is taken by [Dollár et al., 2005].

4. Optical Flow-based features
   The optical flow of a video is a vector field representing the motion of each pixel as a vector of a certain length (the flow-magnitude) and an angle. If such motion information is used rather than appearance the system becomes more robust against changes in appearance. Details for optical flow will be given in Section 3.1.

If grouped by the mid-level a more meaningful categorization can be established:

1. Non Parametric Methods
   The earliest attempts of action recognition fall into this category they use 2D-templates as a representation of a whole action. In [Polana and Nelson, 1997] such 2D-representations are calculated by extracting silhouettes by foreground-background-subtraction. These are aggregated by calculating a so called *Motion Energy Image* which is an addition of all the silhouettes created by an

action and *Motion History Images* which are the result of summing up the silhouettes each multiplied with a temporal decaying weight. These are only feasible for very simple actions.
[Yilmaz and Shah, 2005] describe an algorithm where actions are represented by 3D-objects introduced by stacking 2D contours of the original silhouettes. In this category are also the manifold learning methods which oppose the curse of dimensionality. The curse of dimensionality refers to the fact, that the more dimensional a feature is the sparser its feature space. This is an exponential relation. If an algorithm can find the manifold and express the data within a feature space aligned to this manifold the classification results can benefit. Therefore numerous dimensionality reduction algorithms have been introduced. The *Principle Component Analysis* (PCA) (introduced in [Pentland et al., 1991]) and the *orthogonal Non Negative Matrix Factorization* (NMF) for unusual activity recognition) are two examples. Both of them are explained in Section 3.4. Recognition algorithms like *Suport Vector Machines* can be computed more efficiently after such dimension reduction.

2. Volumetric Methods
   Volumetric methods are methods which consider the video as a three dimensional entity (x, y, t) and evaluate on this structure as a whole. Within this category are filter bank approaches which generally use filters (like oriented Garbor filters or Difference of Gaussians filters) as well in the time as in the space domain to create histograms ([Zelnik-Manor and Irani, 2001], [Zhong et al., 2004])
   In this category also fall part-based approaches where for example a 3D version of the Harris-Corner detector is used to identify points of interest within the spatio-temporal-volume to calculate a feature in its local neighbourhood [Laptev, 2005].

3. Parametric Methods
   Parametric Methods are methods which model the action by a statistical model for example a *Hidden Markov Model* (HMM) or more complex derivatives like in [Kale et al., 2004]. These models are able to describe complex actions and have proven to be feasible for repeating and looping actions. Such HMMs mimic the occurrence of states without knowing them (they are hidden). They assume the occurrence of data to be a product of a Markov-Process.

These structures do not have hard boundaries, their purpose is to cover all of the approaches. The given approach is a non parametric method which uses optical flow and other appearance-based features without performing any background subtraction or identification of geometric entities.

Chapter 3

Theoretical Background

This chapter contains a listing of the theoretical background needed to create the proposed action recognition system. The order of the chapters is inspired by the chain of computations which has to be performed for such a system.

According to the action recognition strategy presented in Chapter 1.4 the task can be divided into subtasks:

1. **Feature Extraction** Section 3.1 is introducing the optical flow chapters 3.3 and 3.2 are introducing the features. Which are the *Histogram of Orientations* (HOG) and the *Local Binary Patterns* (LBP) and their modifications to be computed from the optical flow. Therefore section 3.1 introduces the optical flow. The features are transformed into features of lower dimension by the *Non Negative Matrix Factorization* (NMF). To demonstrate its abilities it is compared to the well known *Principle Component Analysis* (PCA) in Section 3.4.

2. **Region of interest detection** The Support Vector Machine which is employed as Region of Interest detector is subject to Section 3.5.

3. **Prototype Extraction** The *k-Means* algorithm which is utilized to generate frame based prototypes is the focus of Section 3.6.

4. **Prototype Sequence Matching** The prototype sequence matching is based on Dynamic Time Warping (DTW) which is explained in Section 3.7.

3.1 Optical Flow

The optical flow of a video is a vector field which describes the movement in the video during a time interval. According to [Sonka et al., 1999] two conditions have to be satisfied to be able to compute optical flow. The first one is called the Optical Flow Constraint. It specifies the necessity of observed object intensities to be constant over time. Using only this constraint leads to an under-determined equation system. This fact is referred to as *Aperture Problem*. A second constrained is introduced which is called the *Velocity Smoothness Constraint*. It implies that nearby points inside an image do move in a similar manner. An example is given in Figure 3.1.

Figure 3.1: Example of flow field - (left) the two input images (right) the computed flow field between those. For visualization purposes the angle of the field is encoded in the colour value whereas the magnitude in its saturation.

In general there are two types of movements which can not be distinguished by optical flow: The movement of entities inside the camera view field and the movement of the camera itself. Additionally illumination changes in the scene can have strong impact on the flow calculation, since the resulting captured intensities are breaking the optical flow constraint. To reduce the effect of a moving camera a subtraction of the median or mean flow of the whole camera-image is common practice.

Various Optical Flow algorithms have been proposed; one will be described in the following section.

3.1.1 Total Variation L1 Flow

This so called Total Variation L1 flow (TV-L1-Flow) algorithm introduced by [Zach et al., 2007] employs the plain intensity difference between pixels as similarity score. Given a pair of consecutive images at time-step t, which are $\mathbf{X}_0 = \mathbf{X}[t-1]$ and $\mathbf{X}_1 = \mathbf{X}[t]$. The objective is to calculate a displacement field in two dimensions [1] $\mathbf{u} = (u(\mathbf{x}), v(\mathbf{x}))^T$ which minimizes Equation (3.1). This is a definition of the optical flow problem-based on total variation.

$$\int_\Omega \{\lambda\phi\left(\mathbf{X}_0(\mathbf{x}) - \mathbf{X}_1(\mathbf{x} + \mathbf{u}(x))\right) + \psi(\mathbf{u}, \nabla\mathbf{u}, ...)\}d\mathbf{x} \tag{3.1}$$

The first term $\phi\left(\mathbf{X}_0(\mathbf{x}) - \mathbf{X}_1(\mathbf{x} + \mathbf{u}(x))\right)$ represents the image data fidelity and the second $\psi(\mathbf{u}, \nabla\mathbf{u}, ...)$ corresponds to the regularization term which is based on a shape prior. λ is the weighting factor between these two entities: Data fidelity and regularization force.

If $\phi(x) = |x|$ and $\psi(\mathbf{u}, \nabla\mathbf{u}, ...) = |\nabla\mathbf{u}|$,which corresponds to the L1-norm as a distance metric, we can express Equation (3.1) as Equation (3.2).

$$\int_\Omega \{\lambda|\mathbf{X}_0(\mathbf{x}) - \mathbf{X}_1(\mathbf{x} + \mathbf{u}(x))| + |\nabla\mathbf{u}|\}d\mathbf{x} \tag{3.2}$$

The actual difficulty is solving this equation since it is not continuously differentiable. [Zach et al., 2007] accomplish this by linearization of the image intensities, which yields to a convex minimization problem. Such approximation is only valid for small displacements. A coarse-to-fine strategy is required to avoid poor local minima.

They also propose a method of implementation on Graphics Processing Unit (GPU), which performs a near-real-time computation.

[1] $\mathbf{x}$ represents the coordinate vector $\mathbf{x} = (x, y)^T$

3.2 Local Binary Patterns

Local Binary Patterns (LBPs) where developed according to a general model for textures which is based on the local neighborhood of a pixel [Ojala et al., 2000]. They proofed as an outstanding texture descriptor. The original version is shown in Figure 3.2. All of the pixels surrounding a center-pixel are compared to it in a clockwise manner starting from the top left corner. The result of each of this comparisons is stored in the corresponding bit as a binary number. The first bit used is the least significant bit (LSB) the last one the most significant bit (MSB). This resulting binary number represents a descriptor for the texture at this point. It is called LBP-Code further on.

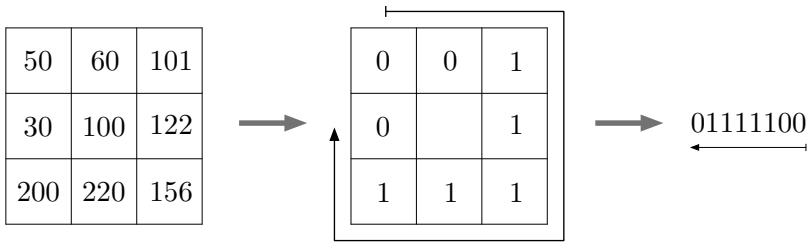


Figure 3.2: Original version of the LBP - All pixels within the 8-neighborhood are compared to the center-pixel in clockwise manner. The result of each of the comparisons is stored in the corresponding bit of a binary number (order: LSB $\rightarrow$ MSB)

Due to the obvious simplicity of *Local Binary Patterns* they lack certain features. They are neither rotation invariant nor adaptive to scale. Therefore several extensions to the general *Local Binary Patterns* where developed.

3.2.1 Circular Local Binary Patterns

Circular Local Binary Patterns disengage form the pixel-grid. The idea is to use a circle as a base structure. The former center-pixel becomes the center of the circle and P points are sampled on a circle with a variable radius r. Due to the fact that P points are sampled on a circle of radius r these sampling-points will generally not be aligned to the pixel-raster their Gray-values have to be interpolated. Usually bilinear interpolation is performed.

Assumed that $I(\mathbf{x})$ is the Gray-value of an image $\mathbf{X}$ at the point $\mathbf{x} = (x, y)^T$ and $\mathbf{x} \in \mathbb{R}^2$ Equation 3.3 presents a mathematical definition of the LBPs.

$$\begin{aligned} \mathrm{LBP}_{P,r}(\mathbf{x}_c) &:= \sum_{j=0}^{P-1} \mathrm{s}\left(\mathbf{X}(\mathbf{x}_j) - \mathbf{X}(\mathbf{x}_c)\right) \cdot 2^j \\ \text{with } \mathrm{s}(x) &:= \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \\ \text{and } \mathbf{x}_j &= \left(x_c + r \cdot \cos(2\pi \tfrac{j}{P}), y_c - r \cdot \sin(2\pi \tfrac{j}{P})\right)^T \text{ where } 1 \leq j < P \end{aligned} \tag{3.3}$$

Uniformity

With advanced use of *Local Binary Patterns* in computer vision it turned out that certain codes have a greater significance than others. They are called uniform. These codes differ from the rest of the codes by their sum of their $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions within the code.

Especially the codes with ≤ 2 of these transitions turned out to be of great significance. They are called 2-uniform.

Rotation Invariance

In several applications rotation invariance is of importance. To reach rotation invariance the original LBP-code is rotated $P-1$ times using a circular bitwise right rotate operation (denoted as Rotate Right operator ROR). The minimum of this P shifted versions is considered to be the LBP-code further on (see Equation (3.4)).

$$\mathrm{LBP}^{ri}_{P,r}(\mathbf{x}_c) = \min_{i=0}^{P-1} \{\mathrm{ROR}(\mathrm{LBP}_{P,r}(\mathbf{x}_c), i)\} \tag{3.4}$$

Where ROR denotes the circular bitwise rotation to the right operator. $\mathrm{ROR}(b, i)$ rotates a binary number b by i bits to the right.

In practice uniformity as well as rotation invariance (and their combination) is accomplished by using a mapping to translate the general LBP-codes into their specialized version. This mapping can be calculated in advance and is usually realized as a look up table. Therefore all of the codes which are not 2-uniform are mapped to one single value which is the first non 2-uniform value available (see Figure 3.3).

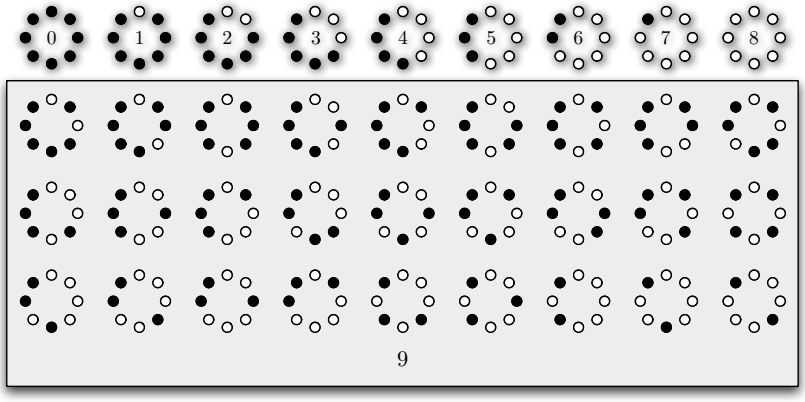


Figure 3.3: 2-uniform rotation invariant LBPs - All of the rotation invariant binary codes for $P = 8$. Black represents a zero and white a one in the binary code. The first row shows the nine 2-uniform rotation invariant codes. The last three rows are coded with the value 9.

For 2-uniform rotation invariant *Local Binary Patterns* a simplification which is shown in Equation (3.5) is possible.

$$\mathrm{LBP}^{riu2}_{P,r}(\mathbf{x}_c) = \begin{cases} count\_ones(\mathrm{LBP}_{P,r}(\mathbf{x}_c)) & \text{if } \mathrm{LBP}_{P,r}(\mathbf{x}_c) \text{ is two-uniform} \\ P+1 & \text{else} \end{cases} \tag{3.5}$$

LBP-Histograms

If the texture for a point at the coordinate vector $\mathbf{x}$ is assumed to be a local probability distribution around it, it is reasonable to create histograms to represent them.

Therefore histograms of LBP-codes in a local neighborhood[2] of a point $\mathbf{x}$ are calculated. These are usually normalized in a way that the sum of all of their bins equals one.

Local Binary Patterns of Flow Magnitude

To use the LBP-descriptor for flow fields the appearance (the original image) is replaced by the flow fields magnitude. The remaining calculation stays the same. Such descriptor will be referred to as LBFP-descriptor further on. While the standard LBP descriptor is calculated on a single image the flow version is dependent on two consecutive images, hence the flow calculation itself requires these two images.

3.3 Histograms of Oriented Gradients

Since [Dalal and Triggs, 2005] proposed the *Histograms Of Gradients* (HOG) descriptor as a superior feature for human detection it was found in many publications considering human detection or action recognition like [Wang et al., 2009] or [Roth et al., 2009]. Their roots can be found in the so called edge orientation histograms and the recently in machine vision very effective Scale Invariant Feature Transform (SIFT) (see [Lowe, 2004], [Nistér and Stewénius, 2006]). While the HOG-descriptor is a dense descriptor its sparse counterpart would be the SIFT. HOGs basic idea is to evaluate well-normalized local histograms of image gradients in a dense grid. The observation that local object appearance can be characterized by the distribution of edge directions without knowing the precise position or gradient is exploited by the HOG-descriptor.

A HOG is calculated in four essential steps (see Figure 3.4): The gradients $\frac{d}{dx}$ and $\frac{d}{dy}$ are computed. Then their orientation φ and magnitude m are computed. The orientations vote into cell histograms, the cell histograms are normalized by incorporating the neighboring cells.

Magnitudes and Orientations

Input images, for simplicities sake assumed to be gray-scale, gradients are calculated by convolution with two 1D forward differences resulting in dx and dy per pixel.

[2] A neighborhood is realized as a block of pixels surrounding $\mathbf{x}$.

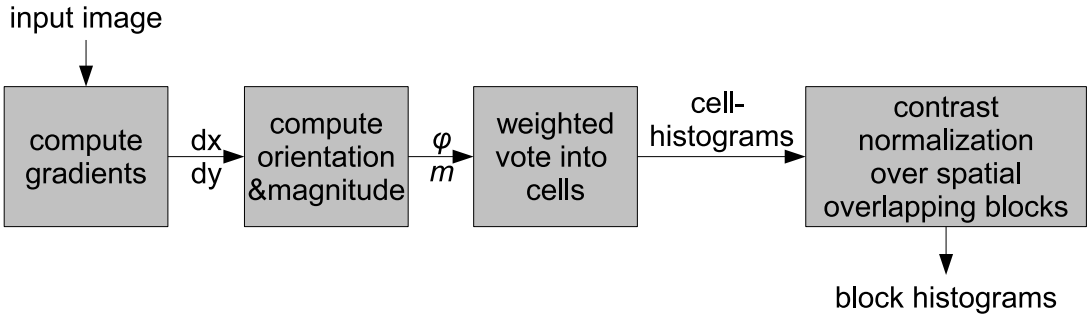


Figure 3.4: Block Diagram of HOG calculation - An overview of the HOG feature extraction presenting its different steps.

Their magnitude is calculated by geometric distance

$$m = \sqrt{dx^2 + dy^2)} \tag{3.6}$$

There are two ways of expressing orientation: The unsigned gradient which results in an angle of $0°$ to $180°$ and the signed gradient resulting in $0°$ to $360°$. Equation (3.7) shows the definition for an unsigned gradient orientation whereas Equation (3.8) shows the signed one.

$$\begin{aligned} \varphi_{unsigned}(dx, dy) &: \mathbb{R}^2 \mapsto [0, 180°) \\ \varphi_{unsigned}(dx, dy) &= \tfrac{180°}{\pi} \left(\arctan\left(\tfrac{dy}{dx}\right) + \tfrac{\pi}{2} \right) \end{aligned} \tag{3.7}$$

$$\begin{aligned} \varphi_{signed}(dx, dy) &: \mathbb{R}^2 \mapsto [0, 360°) \\ \varphi_{signed}(dx, dy) &= \tfrac{180°}{\pi} \left(\operatorname{atan2}\left(dy, dx\right) \right) + \pi) \end{aligned} \tag{3.8}$$

[Dalal and Triggs, 2005] found the unsigned version to be more distinctive for the task of human action detection.

Cell Histograms

The idea is to formulate the local distribution of orientations as histograms, the cell histograms (further on denoted as H_c). A cell refers to a small usually quadratic region of an image. Within this region all orientations are collected in the same cell histogram. If an image is partitioned in such cells it will be referred to as *cell-grid* further on. H_c can be indexed by its bin ($H_c(b)$ with $b \in \mathbb{N}$) and the value within one of these bins equals the sum of weighted magnitudes within this cell which have an orientation corresponding to this bin.

Computing histograms of such continuous values corresponds to a quantization. To minimize the effect of such quantization a weighting of the magnitude according to the orientation is performed. Therefore a magnitude m corresponding to an orientation φ contributes to two histogram bins (see Figure 3.5).

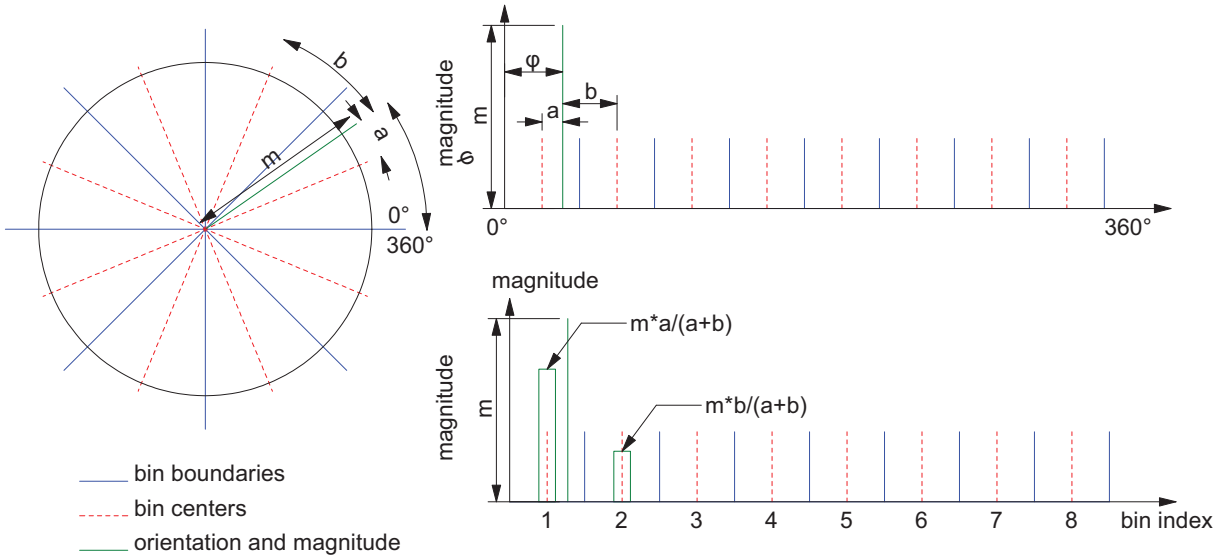


Figure 3.5: HOG magnitude weighting according to orientations - An example of weighting a single pixel's gradient magnitude (length of green line m) according to its orientation (angle of green line ϕ). Left: the orientation falls naturally between bin centers (dashed red). The angular difference of the orientation and the closest bin centers are a and b. Top right: The unwrapped circle. Bottom right: the assignment of the weighted magnitude according to its orientation to discrete bins of the histogram. The green bars represent the final weighted magnitudes in a discrete histogram. (This example assumes an oriented gradient and an eight bin histogram.)

Block Histograms

To gain invariance to illumination and shadowing a contrast normalization of the responses inside the cell histograms is performed. Therefore additionally to the grid of cells, blocks are introduced. A block consists of a number of cells, and blocks are overlapping (see Figure 3.6). The idea behind is to accumulate a measure of local histogram energy over a conjunct larger region to normalize all of the cells inside the block with it.

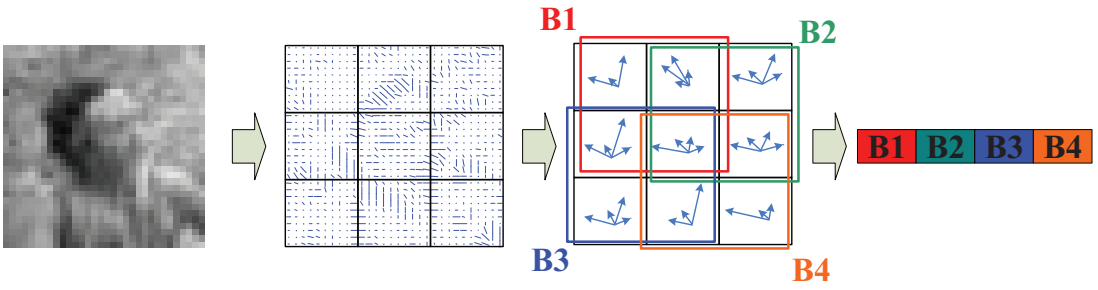


Figure 3.6: Scheme for normalizing overlapping HOG blocks - The gradients of the input image (first image) according to cells (the grid in the second image) are accumulated into histograms per cell (third image) according to their magnitude and orientation (as visualized in figure 3.5). Each cell is normalized by each block (B1 to B4) and the concatenated feature is the result (most right). In this example a block consists of 2×2 cells. (modified illustration from [Zhang and Nevatia, 2008])

Each cell is normalized by each block. Figure 3.6 shows that a single cell (the middle one) is normalized four times for a 2×2 cells per block configuration. The conjunction of all this different normalizations

of a cell will be referred to as a *HOG-descriptor* further on.

3.3.1 Histogram of Flow Orientation

A Histogram of Flow Orientation (HOF) [Laptev et al., 2008] is an adaptation of the HOG principles to apply them on flow fields instead of image gradients. Here the orientations are the orientations of the flow field. Signed orientations (Equation 3.8) are used which results in a range of 0°to 360°which reflects the important information of the current direction of the flow-field. The flow-fields magnitude m is again computed by Equation 3.6) where dx and dy are the flow-fields displacement ($\frac{dx}{dt}$ and $\frac{dy}{dt}$).

3.4 Dimensionality Reduction PCA versus NMF

Since the calculated features are of high dimension a dimensionality reduction is performed to be able to perform all following computation steps faster and more memory efficient.

The motivation for dimensionality reduction or sub-space-methods is that high-dimensional data has the general property that data points lie close to the manifold of a much lower dimensionality [Bishop, 2007].

If $\mathbf{V} \in \mathbb{R}^{N \times M}$ denotes the data points, where each of the M columns corresponds to an N-dimensional observation. A linear dimensionality reduction can be expressed by a linear transformation into a different vector space of dimension $L \leq N$.

Such can alternatively be expressed as a factorization of the original data $\mathbf{V}$ into two matrices $\mathbf{W}$ and $\mathbf{H}$ such that an approximation of $\mathbf{V}$ is reached (see Equation (3.9)) where $\mathbf{W}$ constitutes a base and $\mathbf{H}$ contains the coefficients (see [Lee and Seung, 1999]).

$$\mathbf{V}_{(N \times M)} \approx \mathbf{W}_{(N \times L)} \, \mathbf{H}_{(L \times M)} \tag{3.9}$$

Two such sub-space methods presented are the *Principle Component Analysis* (PCA) and the *Non Negative Matrix Factorization* (NMF). The difference lies within the constraints for the factorization.

3.4.1 Principle Component Analysis

Principle Component Analysis (PCA) also called *Karhunen-Loève-transformation* is a method which transforms a vector space which contains correlated observations (in our case features) into less correlated variables which are called *principle components*. According to [Hotelling, 1933] such projection maximizes the variance of data while minimizing the sum of squares of the projection errors (pointed out in [Pearson, 1901]).

The *eigenvectors* corresponding to the L greatest *eigenvalues* of the *covariance matrix* are the so called *principal components*. A famous application are the so called *eigenfaces* used for facial recognition in [Pentland et al., 1991]. Eigenfaces are created by using a concatenation of the intensity values of an image as features. This simple feature allows a visual representation of the principal components. As each principal component and eigenvector have the same size as the features, they can be shown as images themselves. In respect to Equation (3.9) the columns of $\mathbf{W}$ correspond to the concatenation of the mean face and the $L - 1$ eigenfaces.

A linear combination of the mean face and these eigenfaces weighted by the coefficients in $\mathbf{H}$ yields to an approximation of the data in $\mathbf{V}$. A visual representation is shown in Figure 3.7. The columns of $\mathbf{W}$ will be referred to as base-images further on.

There is no obvious interpretation for the eigenfaces, they lack intuitive meaning, but one can notice that each eigenface contains information for each location of the image. Therefore they are referred to as a holistic sub-space method [Lee and Seung, 1999].

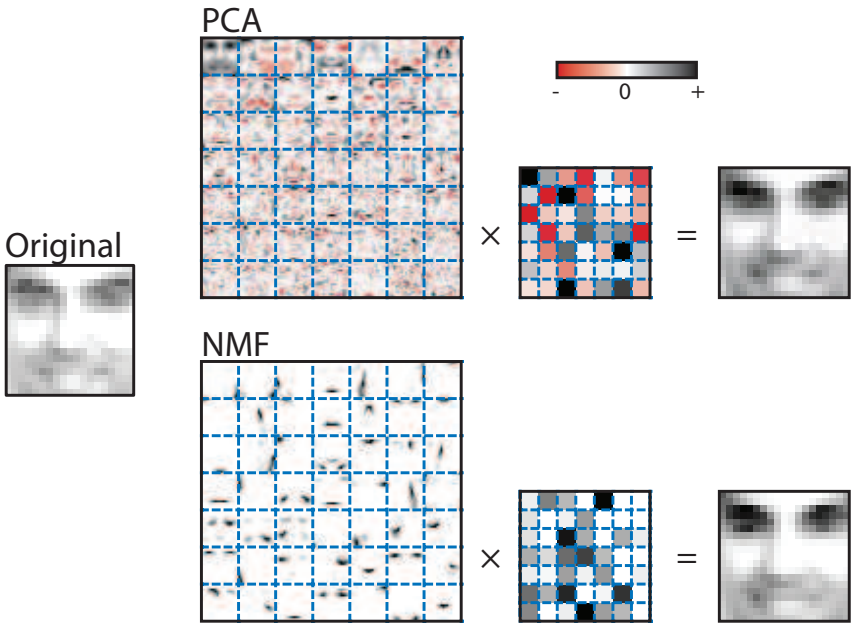


Figure 3.7: Comparison of basis-images for NMF and PCA - (left) the original image $\mathbf{V}$. (right) The superposition of the base-images (columns of $\mathbf{W}$) to approximate $\mathbf{V}$ by $\mathbf{WH} = \tilde{\mathbf{V}}$ for PCA and NMF. PCA allows negative values (red). Therefore the PCA base-images are a holistic representation. NMF only allows positive values (black). Therefore the base-images are sparse. Significant parts like eyes or mouth can be identified within the set of base images. (adapted from [Lee and Seung, 1999])

3.4.2 Non Negative Matrix Factorization

The *Non Negative Matrix Factorization* (NMF) on the other hand is a part-based sub-space method. The constraints for the factorization are that $\mathbf{H}$ and $\mathbf{W}$ are non negative. Therefore a subtraction is not permitted. Only additive linear superpositions of the base images are possible. For this reasons intuitive notion of combining parts to build a whole is accomplished. The base-images (columns of $\mathbf{W}$) are sparse, most of the elements are zero.

From a more general perspective NMF can also be seen as model for the generation of visible variables $\mathbf{V}$ from the hidden variables $\mathbf{H}$ where each hidden variable activates a subset of visible variables [Lee and Seung, 1999]. This properties have made NMF popular in computer vision tasks, as well as in document analysis. [Roth et al., 2009] built a human action recognition system using NMF factorization to reduce the dimensionality of HOG and HOF features before classification.

Formally an NMF factorization of a matrix $\mathbf{V} \in \mathbb{R}_{+}^{(N \times M)}$ into the two matrices $\mathbf{W} \in \mathbb{R}_{+}^{(N \times L)}$ and $\mathbf{H} \in \mathbb{R}_{+}^{(L \times M)}$ with $L < N$ can be described as an optimization problem of the form

$$\min \|\mathbf{V} - \mathbf{WH}\|^2 \text{ where } \mathbf{W} \geq 0 \text{ and } \mathbf{H} \geq 0 \tag{3.10}$$

and $\|.\|^2$ denotes the squared euclidian norm.

The computation of such an NMF factorization can not be performed as a closed-form solution, therefore an iterative approach for estimating $\mathbf{V}$ and $\mathbf{W}$ has to be performed. The numerical methods to accomplish solving such iterative minimization problems vary in literature and are explained in detail in [Lee

and Seung, 1999]. One method is called the iterative multiplicative update method. The idea is to decompose the error gradient ∇E into a positive and negative part where $\nabla E = (\nabla E)^- - (\nabla E)^+$ where $(\nabla E)^+ > 0$ and $(\nabla E)^- > 0$. This leads to an updating rule for each element Θ_{ij} of both matrices ($\mathbf{W}$ and $\mathbf{H}$) described by

$$\Theta_{ij} \leftarrow \Theta_{ij} \frac{[(\nabla E)^-]_{ij}}{[(\nabla E)^+]_{ij}} \tag{3.11}$$

The update rules for the matrix elements w_{ij} of $\mathbf{W}$ and h_{kl} of $\mathbf{H}$ are therefore

$$w_{ij} \leftarrow w_{ij} \frac{[\mathbf{V}\mathbf{H}^T]_{ij}}{[\mathbf{W}\mathbf{H}\mathbf{H}^T]_{ij}} \tag{3.12}$$

$$h_{kl} \leftarrow h_{kl} \frac{[\mathbf{W}^T\mathbf{V}]_{kl}}{[\mathbf{W}^T\mathbf{W}\mathbf{H}]_{kl}} \tag{3.13}$$

This update scheme is applied multiple times until the change of $\mathbf{H}$ and $\mathbf{V}$ is below a certain threshold or a maximum number of iterations is reached.

3.4.3 Orthogonal Non Negative Matrix Factorization

If additionally to the non negativity constraint the constriction of orthogonality of $\mathbf{W}$ is imposed, a slightly different minimization problem arises (see Equation (3.14)).

$$\min\|\mathbf{V} - \mathbf{W}\mathbf{H}\|^2 \text{ where } \mathbf{W} \geq 0 \text{ and } \mathbf{H} \geq 0 \text{ and } \mathbf{W}^T\mathbf{W} = \mathbf{I} \tag{3.14}$$

If orthogonality of $\mathbf{W}$ is given it is possible to calculate the coefficients by [3]:

$$\mathbf{H} = \mathbf{W}^+\mathbf{V} \approx \mathbf{W}^T\mathbf{V} \tag{3.15}$$

$$\mathbf{H} = \mathbf{P}\mathbf{V} \text{ with } \mathbf{P} = \mathbf{W}^\mathbf{T} \tag{3.16}$$

since this is a projection we will refer to $\mathbf{P}$ as NMF-projection-matrix further on.

Nevertheless orthogonality of W is not only feasible it also outperforms standard NMF in several experiments since a more localized part-based representation can be found [Choi, 2008].

To solve the optimization problem given in Equation (3.14) the so called *true gradient* is used which preserves $\mathbf{W}^T\mathbf{W} = \mathbf{I}$. Therefore the decomposition of the error gradient is performed by

$$\begin{aligned} \tilde{\nabla}_W E &= -\mathbf{W}[\nabla_W E]^T\mathbf{W} &&+\nabla_W E \\ &= (\nabla_W E)^+ &&-(\nabla_W E)^- \\ &= (\mathbf{W}\mathbf{H}\mathbf{V}^T\mathbf{W}) &&-(\mathbf{V}\mathbf{H}^T) \end{aligned} \tag{3.17}$$

[3] $\mathbf{W}^+$ denotes the Moore–Penrose pseudo-inverse of $\mathbf{W}$.

The resulting update scheme according to Equation (3.11) for $\mathbf{W}$ is therefore

$$w_{ij} \leftarrow w_{ij} \frac{[\mathbf{V}\mathbf{H}^T]_{ij}}{[\mathbf{W}\mathbf{H}\mathbf{V}^T\mathbf{W}]_{ij}} \tag{3.18}$$

whereas the update sheme for $\mathbf{H}$ stays the same (Equation (3.13)).

3.5 Support Vector Machines

Support Vector Machines (SVMs) are widely used non-probabilistic binary classifiers. They model the decision boundary as a hyperplane between data-points of two classes within a feature-space, which is why they are a decision machine. They are a *maximum margin classifier* which refers to the fact that such a decision boundary is chosen in a way that the distance to the first data-points of each (of the two) classes to the decision boundary is maximized [Bishop, 2007].

A two class classification problem (C={-1,+1}) using a linear model can be defined as

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{3.19}$$

where $\mathbf{x}$ denotes a single input vector. $\mathbf{w}$ is called a weight vector. $\phi(\mathbf{x})$ is a fixed not necessarily linear feature space transformation and b is called the bias parameter. $\mathbf{w}^T$ determines the orientation of the decision surface whereas b determines its location. An input vector $\mathbf{x}$ is classified as $C = -1$ if $y(\mathbf{x}) \geq 0$ as $C = 1$ otherwise. The training data set consists of N input vectors $\mathbf{x}_1 ... \mathbf{x}_N$ and a corresponding target vector containing the class labels $t_1 ... t_N \in C$. Therefore the product $t_n y(\mathbf{x}_n) > 0$ for all data-points.

The aim of training with this dataset is to determine the hyperplane ($\mathbf{w}$,b) which maximizes the margin. The normal distance of a data point $\mathbf{x}$ from the hyperplane is given by $\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$. If we are only interested in solutions where all the data-points are correctly classified ($t_n = \text{sign}\,(y(\mathbf{x}_n))$) the distance of a decision point is given by Equation (3.20). Such is only possible for a linear separable problem.

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|} \tag{3.20}$$

The maximum margin solution is given by

$$\underset{\mathbf{w},b}{\text{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \min_{n \in [1,N]} \left(t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) \right) \right\} \tag{3.21}$$

As the distance measure of a data-point $\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$ leaves one degree of freedom we can substitute (without loss of generality) $\mathbf{w} \rightarrow k\mathbf{w}$ and $b \rightarrow kb$ such that the closest point has a distance of 1. Therefore $t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) \geq 1$ is valid for all samples, which leads to

$$\min_{n \in [1,N]} \left(t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) \right) = 1. \tag{3.22}$$

Therefore Equation (3.21) is simplified to its canonical representation

$$\begin{aligned} &\underset{\mathbf{w},b}{\operatorname{argmax}} \tfrac{1}{\|\mathbf{w}\|} = \underset{\mathbf{w},b}{\operatorname{argmin}} \|\mathbf{w}\| = \underset{\mathbf{w},b}{\operatorname{argmin}} \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &\text{with respect to } t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) \geq 1 \end{aligned} \tag{3.23}$$

Such an optimization problem can be solved by the use of *Lagrange* multipliers. However in general a problem is not linear separable therefore some of the data points will be on the wrong side of the hyperplane. Such points are represented in the optimization problem by a penalty term for each of them as described in [Cortes and Vapnik, 1995].

These penalty terms shall be defined by $\xi_n = |t_n - y(\mathbf{x}_n)|$ for points which are at the wrong side of the hyperplane and $\xi_n = 0$ for all correct data-points. Therefore the optimization problem can be described as

$$\begin{aligned} &\underset{\mathbf{w},b}{\operatorname{argmin}} \left\{ K \textstyle\sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \right\} \\ &\text{subject to } t_n y(\mathbf{x}_n) \geq 1 - \xi_n \text{ for all } n \in [1, N] \end{aligned} \tag{3.24}$$

where $K > 0$ controls the trade-off between data fidelity and generalization.

The solution of this problem is found by introducing the N nonnegative Lagrange Multipliers $\mathbf{a} = a_1 ... a_n$ and solving the dual problem which according to [Chang and Lin, 2011] is defined as

$$\begin{aligned} \underset{\mathbf{a}}{\operatorname{argmin}} \quad & \left\{ \tfrac{1}{2}\mathbf{a}^T Q \mathbf{a} - \mathbf{e}^T \right\} \\ \text{subject to} \quad & \mathbf{t}^T \mathbf{a} = 0 \\ & 0 \leq a_n \leq K,\ n = 1...N \end{aligned} \tag{3.25}$$

where $\mathbf{e} = [1, ..., 1]^T$ and $Q_{ij} = t_i t_j k(x_i, x_j)$ and the kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_t)^T \phi(\mathbf{x}_j)$.

In the original feature space such decision boundary is a hyperplane only if $\phi(\mathbf{x})$ is a linear function. Therefore complex non-linear boundaries can be expressed by increasing the dimensionality of the picture of ϕ. The usage of the kernel k is an efficient method for avoiding to perform calculations in this higher dimensional space. In literature this is referred to as the *kernel trick*.

Once such problem is solved the optimal $\mathbf{w}$ is given by

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x_n}) \tag{3.26}$$

It can be observed that points which fulfill $[t_n(\mathbf{w}^T\mathbf{x}_n + b) - 1 + \xi_n] > 0$ do not mater because the corresponding a_n is 0 which reflects the fact that they are no support vectors.

Due to the fact that only a small number of a_n are non-zero this representation is a sparse one. The x_n corresponding to the $a_n \neq 0$ are the support vectors. They lie on the margin which is expressed by

$$t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) = 1$$

Therefore b can be calculated by any support vector[4] $\mathbf{x}_n$ as

$$b = 1/t_n - \mathbf{w}^T\phi(\mathbf{x}_n) = t_n - \mathbf{w}^T\phi(\mathbf{x}_n). \tag{3.27}$$

The final decision function is given by

$$\operatorname{sign}\left(\mathbf{w}^T\phi(\mathbf{x}) + b\right) = \operatorname{sign}\left(\sum_{n=1}^{N} t_n a_n k(\mathbf{x}_n, \mathbf{x}) + b\right) \tag{3.28}$$

Linear Support Vector Machines

In case of linear SVMs we can simplify the optimization problem since $\phi(\mathbf{x}) = \mathbf{x}$ therefore $k = \phi(\mathbf{x})^T\phi(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$.

Therefore the decision function of equation (3.28) is simplified to

$$\begin{aligned} \operatorname{sign}\left(y(\mathbf{x}_{test})\right) &= \operatorname{sign}\left(\sum_{n=1}^{N} t_n a_n \mathbf{x}_n^T \mathbf{x}_{test} + b\right) \\ &= \operatorname{sign}\left(\mathbf{a}^T \operatorname{diag}(\mathbf{t})\mathbf{X}^T\mathbf{x}_{test} - b\right) \\ &= \operatorname{sign}\left(\mathbf{A}\mathbf{x}_{test} - b\right) \end{aligned} \tag{3.29}$$

where $\mathbf{x}_{test}$ is a single test sample to be classified and $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]$ is the concatenation of all training vectors while $\mathbf{A} = \mathbf{a}^T \operatorname{diag}(\mathbf{t})\mathbf{X}^T$. $\mathbf{A}$, which has only one row, and b are therefore expressing the whole functionality of the linear SVM.

Scaling of Input Data

It is common to perform a scaling for the input vectors for SVMs such that each of their dimensions has a range either of $[-1, 1]$ or $[0, 1]$. This, usually, results in better separability and less numerical problems. The most simple method is referred to a *maximum normalization* and is a devision of each feature dimension by the maximum value which occurred in the training set.

Assume a linear SVM trained on such maximum normalized features $\acute{\mathbf{x}} = diag(\frac{1}{\mathbf{m}})\mathbf{x}$ with the maximas $\mathbf{m} = \max_{rows}(\mathbf{X})$ which results in different a_n which are denoted as $\acute{a}_n$ and a different b denoted as $\acute{b}$, then the decision-function of equation (3.29) is changed to be

$$\begin{aligned} \operatorname{sign}\left(y(\mathbf{x}_{test})\right) &= \operatorname{sign}\left(\acute{\mathbf{a}}^T \operatorname{diag}(\mathbf{t})\acute{\mathbf{X}}^T\acute{\mathbf{x}}_{test} - \acute{b}\right) \\ &= \operatorname{sign}\left(\acute{\mathbf{a}}^T \operatorname{diag}(\mathbf{t})\acute{\mathbf{X}}^T \operatorname{diag}(\tfrac{1}{\mathbf{m}})\mathbf{x}_{test} - \acute{b}\right) \\ &= \operatorname{sign}\left(\acute{\mathbf{A}}\mathbf{x}_{test} - \acute{b}\right) \end{aligned} \tag{3.30}$$

with $\acute{\mathbf{A}} = \acute{\mathbf{a}}^T \operatorname{diag}(\mathbf{t})\acute{\mathbf{X}}^T \operatorname{diag}(\frac{1}{\mathbf{m}})$.

[4] In practice a stable solution is given by averaging all support vectors.

3.6 k-Means

Many of the existing publications in the field of action and activity recognition make extensive use of so called *prototypes*. A prototype is a representative for a group of elements. For example [Lin et al., 2009] use pose prototypes for action recognition while [Nater et al., 2010] incorporate them for unsupervised activity analysis.

k-Means is a clustering algorithm which can be employed to generate such prototypes. Its objective is to find K clusters within a data set $\{\mathbf{x}_1, \mathbf{x}_n\}$ which minimize

$$\begin{aligned} J &= \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \\ &\text{with } r_{nk} \in \{1, 0\},\ r_{nk} = 1 \Rightarrow r_{nj} = 0 : \forall j \neq k \end{aligned} \tag{3.31}$$

where $\boldsymbol{\mu}_k$ is the cluster mean and $r_{nk} = 1$ selects a cluster k for data point n. The goal is to find the $\{r_{nk}\}$ and $\{\boldsymbol{\mu}_k\}$ which minimize J.

To solve such problem an EM (expectation maximization) method is employed. In the expectation step each data point $\mathbf{x}_n$ is assigned to the closest cluster k which minimizes $\|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$.

In the maximization step the new means $\boldsymbol{\mu}_k$ are calculated by

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} r_{nk} \mathbf{x}_n}{\sum_{n=1}^{N} r_{nk}}. \tag{3.32}$$

The algorithm is initialized with random cluster assignments (random r_{nk}). After some iterations it converges to a local minimum. Therefore the algorithm is usually run several times to find a good solution.

3.6.1 Hierarchical k-Means

The idea behind hierarchical k-Means is to create a tree of clusters by performing a recursive k-Means calculation for the data assigned to each previously found cluster. This leads to a tree T of clusters with each node containing a mean value.

Once such tree has been found a new data sample (which was not used for generating the tree) can be assigned to a specific leaf by depth first search, where at each node of the tree the child which is the nearest neighbor (its mean) to the sample is found and the corresponding node is visited. At some point a leaf node is reached. If the tree is well developed (covers the complexity of the data sufficiently) its leaf nodes are representative prototypes for the data.

Suppose a tree T has been calculated to generate N prototypes (leaf nodes) and a single k-Means has been calculated with $k_2 = N$. Then the number of sample to mean distance calculations in case of T is equal to $depth(T)k_1 = \frac{\log N}{\log k_1} k_1$. while the single k-Means needs N distance calculations. Therefore by usage of such hierarchy a prototype can be calculated much faster since $k_1 \ll k_2$. This also means that with increasing k_1 the tree is less efficient.

Another way to create a similar tree is to use a single k-Means calculation to generate clusters and to build a tree upon these clusters by merging them to a hierarchy (as done in [Lin et al., 2009]).

3.7 Dynamic Time Warping

The *Dynamic Time Warping* (DTW) algorithm became popular for its ability to establish a similarity measure for two time series which minimizes the effects of local stretching in the time axis of either of these. Therefor a quasi elastic transformation in time is performed which is referred to as warping. The fields of application range from data mining, audio analysis and speech recognition, gene sequence analysis to action recognition.

[Lin et al., 2009] recently presented an action recognition system which finds per-frame prototypes and compares prototype sequences to find a corresponding action. The comparison is performed by incorporating DTW as a distance measure.

Assume two time series to be $\mathbf{X} = (x_1, ..., x_N)$ and $\mathbf{Y} = (y_1, ..., y_M)$ which have been sampled at an identical sample rate. $\mathbf{Y}$ is a warped version of $\mathbf{X}$ which means that the sequence of occurrences of new values is the same only the number of time steps per value varies (see Figure 3.8). This means $\mathbf{X}$ and $\mathbf{Y}$ differ only in a localized stretching of the time axis. The DTW distance of such sequences will be 0.

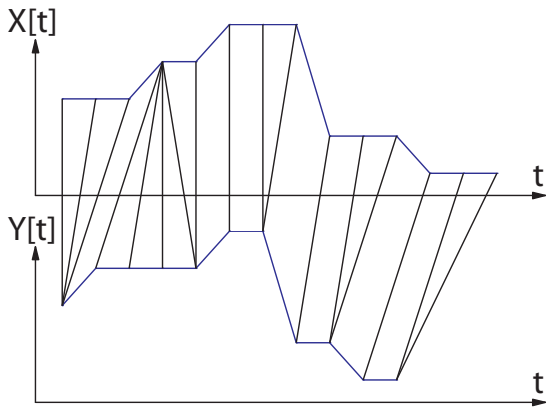


Figure 3.8: Comparison of two time series by DTW - $\mathbf{X}$ and $\mathbf{Y}$ are two time series which only differ in warping. The warping is illustrated by the angle of the black lines which are connecting the two time series. Two equal time series would cause only parallel vertical lines. (inspired by [Chan, 2007])

To solve the problem of finding such warping a dynamic programming approach is taken. Therefore the problem is stated as follows:

Given two time series $\mathbf{X} = (x_1, ..., x_N)$ and $\mathbf{Y} = (y_1, ..., y_M)$ construct a warp path $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_K)$ where $\max(N, M) \leq K < N + M$ and the k^{th} element of $\mathbf{W}$ is $\mathbf{w}_k = (n, m)^T$ where n is the temporal index for $\mathbf{X}$ and m the temporal index for $\mathbf{Y}$.

A warp path has to fulfill following requirements to be an optimal warp path:

R 1: Each index of both time series has to be part of the warp path.

R 2: The n and m of the warp path have to monotonically increase which is why the lines in Figure 3.8 are not crossing. This means for a $\mathbf{w}_k = (n, m)^T$ and $w_{k+1} = (\acute{n}, \acute{m})^T$, $n \leq \acute{n} \leq n + 1$ and $m \leq \acute{m} \leq m + 1$ has to be valid.

R 3: The warp path has to start at the beginning of both time series and to end at the end of both of them. Therefore the first element of the warp path $\mathbf{w}_1$ has to be $(1,1)^T$ whereas the last $\mathbf{w}_k = (N,M)^T$.[5]

The optimal warp path is defined as the warp path $\mathbf{W}$ which minimizes the distance

$$Dist(\mathbf{W}) = \sum_{k=1}^{k=K} dist(\mathbf{X}[w_{1,k}], \mathbf{Y}[w_{2,k}]) \tag{3.33}$$

where $w_{1,k}$ is the first element of $\mathbf{w}_k$ which is indexing $\mathbf{X}$ and $w_{1,k}$ is the second element of $\mathbf{w}_k$ which is indexing $\mathbf{Y}$, therefore $\mathbf{w}_k = (w_{1,k}, w_{2,k})^T$. $dist$ is a distance measure of any kind evaluating to a cost representing the similarity of two data points. Usually an euclidian distance is used.

Dynamic Programming Solution

To solve this problem a two dimensional distance matrix $\mathbf{D} \in \mathbb{R}^{(N \times M)}$ is constructed. Where the value at $\mathbf{D}(n,m)$ is the distance $Dist(\mathbf{W})$ of the optimal warp path of $\mathbf{X}_n = (x_1, ..., x_n)$ and $\mathbf{Y}_m = (y_1, ..., y_m)$ where $n \leq N$ and $m \leq M$. The last value $\mathbf{D}(N,M)$ is the distance of the minimal warp path of $\mathbf{X} = \mathbf{X}_N$ and $\mathbf{Y} = \mathbf{Y}_M$.

To construct such matrix the dynamic programming paradigm is employed: Assume we want to calculate a distance value at $\mathbf{D}(n,m)$ and that all distances of all possible shorter warp paths have already been calculated then

$$\mathbf{D}(n,m) = dist(\mathbf{X}[n], \mathbf{Y}[m]) + \min(\mathbf{D}(n-1,m), \mathbf{D}(n,m-1), \mathbf{D}(n-1,m-1)) \tag{3.34}$$

reflects the fact that to create a one element longer optimal warp path, we have to find the shortest warp path up to this element and add the distance of the current element. This distance is denoted as $dist$ and represents any kind of reasonable distance measure for the data-points. Mostly the euclidean distance is used.

By doing so the distance of the minimum warp path is found. It is the value in $\mathbf{D}(N,M)$ but the path itself is unknown.

If the warp path itself is needed, one has to walk through the distance matrix in reversed order beginning from the last element $\mathbf{D}(N,M)$ and performing a greedy search. As already required by the definition of a warp path the last element $\mathbf{w}_K = (N,M)^T$. The one before is given by evaluating the minimum of $\mathbf{D}(N-1,M)$, $\mathbf{D}(N,M-1)$ and $\mathbf{D}(N-1,M-1)$ and add the corresponding indices to the warp path. If $\mathbf{D}(\mathbf{x}) = \mathbf{D}(n,m)$ with $\mathbf{x} = (n,m)^T$ this can formally be defined for any k as

$$\mathbf{w}_{k-1} = \mathbf{w}_k + \underset{d \in \{(-1,0),(0,-1),(-1,-1)\}}{\operatorname{argmin}} (\mathbf{D}(\mathbf{w}_k + d^T)). \tag{3.35}$$

In such way a complete warping path can be constructed if $\mathbf{D}(n,0)$ evaluates to ∞ for all n and $\mathbf{D}(0,m)$ evaluates to ∞ for all m. This ensures that $\mathbf{D}(1,1)$ is reached.

A visualization of $\mathbf{D}$ and its connection to the two time series $\mathbf{X}$ and $\mathbf{Y}$ plus the resulting warp path $\mathbf{W}$ is given in Figure 3.9.

[5] Such does not have to be the case, if the aim is to find a subsequence match.

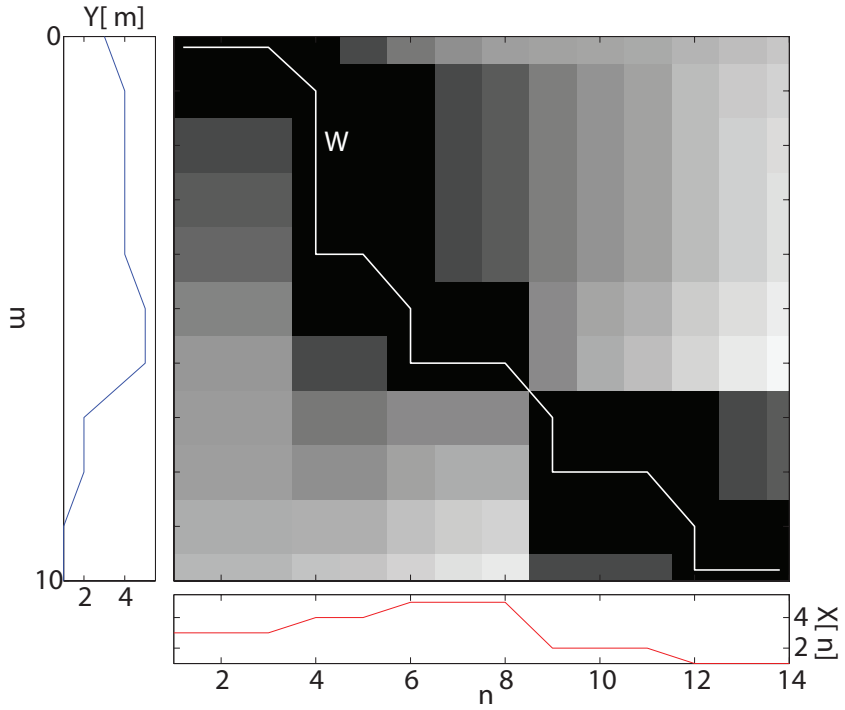


Figure 3.9: A distance matrix and a warping path calculated by DTW - The graphic shows the two time series $\mathbf{X}$ (bottom) and $\mathbf{Y}$ (left) and in the center the data-point to data-point distances calculated by $dist(\mathbf{X}[n], \mathbf{Y}[m])$ for all n and m. Small values of $dist(\mathbf{X}[n], \mathbf{Y}[m])$ are represented as dark regions while large values are bright. Additionally the resulting warping path $\mathbf{W}$ (the white line) is shown. It can be observed that $\mathbf{W}$ is monotone in both directions.

Complexity of DTW

As each cell of $\mathbf{D}$ is filled once the time as well as space complexity equals to $\mathbf{O}(N \cdot M)$. To reduce the time-complexity several methods have been proposed. Two of the most common are the *Sakoe-Chuba Band* and the *Itakura Parallelogram*. Both of them limit the number of cells which have to be evaluated within the cost matrix $\mathbf{D}$ (see Figure 3.10). Cells which are not inside the restricted region have ∞ cost.

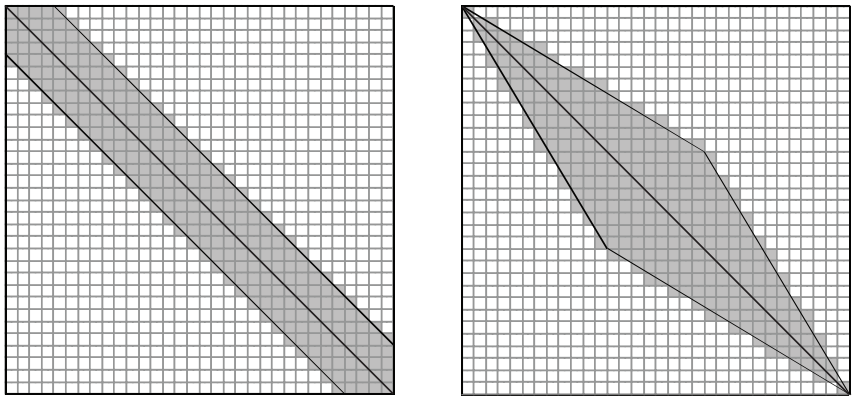


Figure 3.10: Sakoe-Chuba Band and Itakura Parallelogram boundaries - Comparison of the cells which have to be evaluated using the two methods of restriction (left) Sakoe-Chuba Band and (right) Itakura Parallelogram. Light cells are not computed and if accessed they are evaluated to infinite cost.

It turns out that such restrictions do not only reduce the number of operations needed to find a solution they are also beneficial for all problems where the optimal warp path is close to the diagonal of the cost matrix, since single failures within the time series are not compromising the whole result. Other approaches evaluate DTW in a coarse to fine manner where the coarse result constitutes a boundary for the finer one (like [Chan, 2007]). A good overview of this and further techniques is given in [Müller, 2007].

3.7.1 Subsequence matching by Dynamic Time Warping

Given the sequence of prototypes which have been established during the last frames of the image stream, and a number of candidate sequences corresponding to certain actions which have been established during training (as can be seen in Figure 1.4), the problem at hand, is to match a number of prototypes of the query sequence to the whole of prototypes of one of the candidate-sequences. Such is a problem of subsequence matching. Subsequence matching by dynamic time warping differs by releasing one of the requirements **R 3** in Section 3.7: The requirement that the warping path has to start at the beginning of both time series and end at their ends. While the candidate sequence will be completely covered by the DTW-path only part of the query will be included.

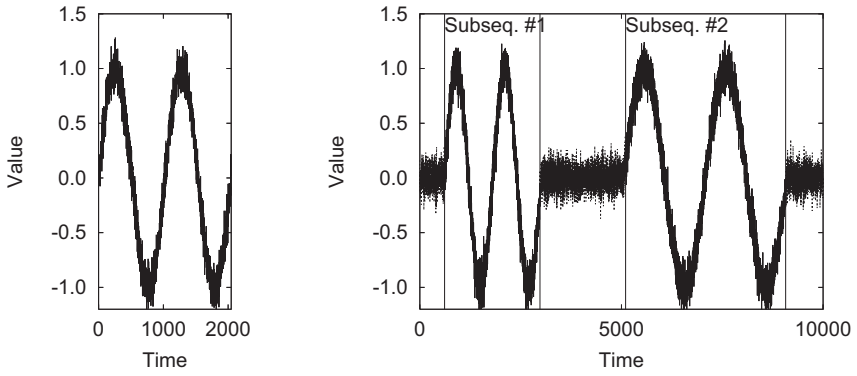


Figure 3.11: Subsequence matching - An illustration of the prototype-stream (right) and the candidate sequence (left) which is searched for by the example of a sine wave. Two subsequence matches could be found in this example [Sakurai et al., 2007].

The naive approach would be to run the standard dynamic programming solution explained above on segments of the candidate sequences but this is infeasible due to its computational cost. Another faster approach (as explained in [Müller, 2007]) for performing subsequence matching by DTW is to calculate a full data-point to data-point distance matrix by dist($\mathbf{X}$[n],$\mathbf{Y}$[m]) and to search local minima in the last column of it (denoted as Δ). Then for each of these minima (denoted as b) a DTW-path is computed according to Equation 3.35. The resulting first element is considered to be the starting time of the subsequence-match, while its last element is the end time . This is visualized in Figure 3.12.

As the computation of the distance matrix as well as the back tracing of all locally minimal paths is still computationally expensive a different algorithm specialized for streamed data was introduced by [Sakurai et al., 2007] and will be introduced in Section 4.8.2.

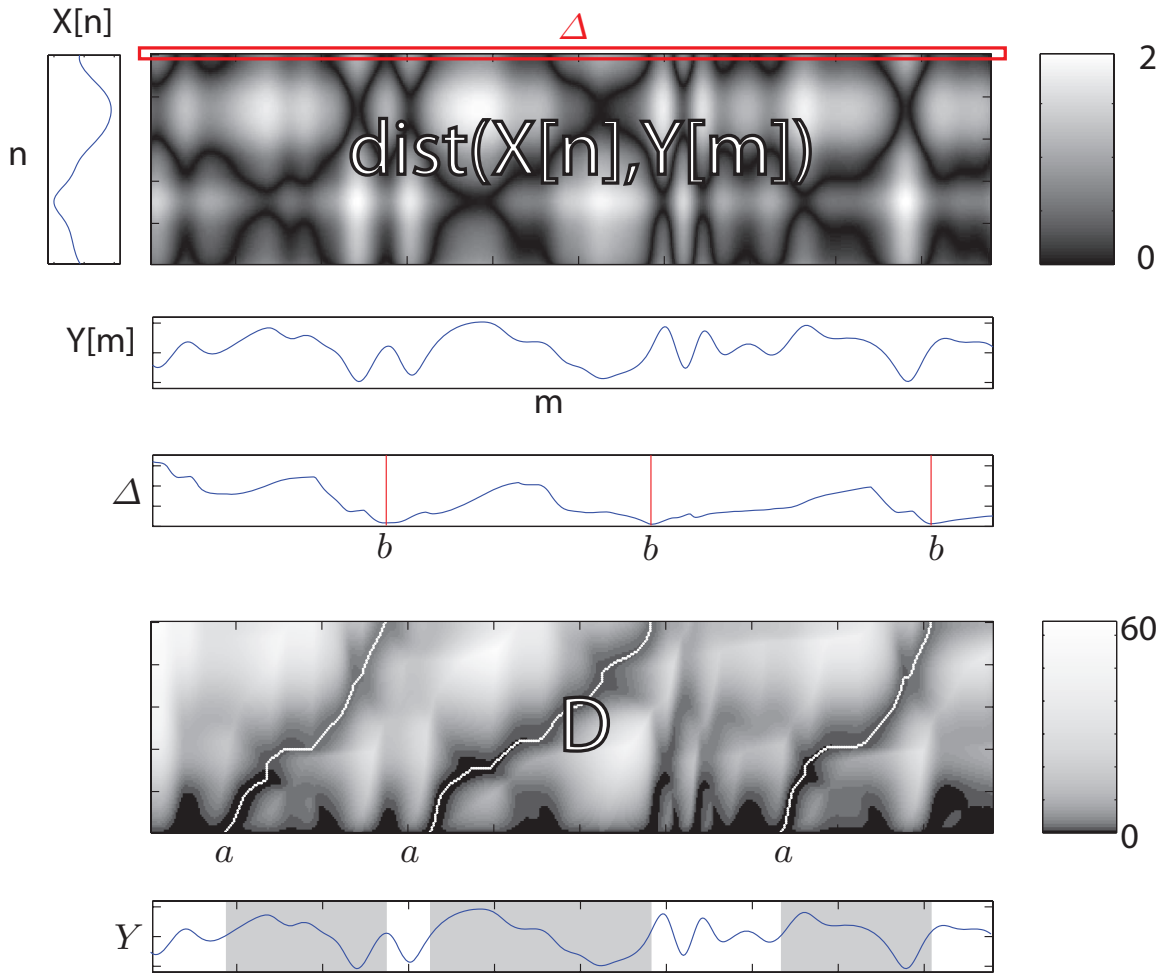


Figure 3.12: Subsequence matching by dynamic time warping - Occurrences of the sequence $\mathbf{X}$ are searched within the sequence $\mathbf{Y}$. Therefore $dist(\mathbf{X}[n], \mathbf{Y}[m])$ is calculated for all n and m. Than the first row of $dist(\mathbf{X}[n], \mathbf{Y}[m])$ (indicated as Δ) is searched for local minima (red lines) each denoted as b. Each b is an end point for a warping path (white lines in $\mathbf{D}$) starting at a. The extend of such warping path is considered a subsequence-match (gray regions in the bottom diagram) (adaption from [Müller, 2007])

Chapter 4

A System for Real-time Action Recognition

To create a real-time action recognition system the two main objectives are to calculate the features in real time and to be able to classify actions by analyzing a unknown number of images captured in the past for their content in an efficient way. The first objective is reached by computing features while exploiting the massively parallel computation abilities of the GPU . The second objective is fulfilled by an appropriate system design and algorithmic concept. The system design is discussed in section 4.1 while section 4.2 deals with the necessary preparations to run such a system - the training phase. In section 4.3 the basics for feature computation on the GPU and the resulting design decisions will be discussed while section 4.4 and 4.5 describe the implementation of these features. Section 4.6 deals with finding the correct scale and the computation of a single feature per frame while section 4.7 explains the generation of a prototype per frame and their concatenation resulting in prototype sequences.

An additional requirement is to find the end of a performed action also referred to as repetition recognition. This is a requirement which results from the target application as game interface. Multiple executions of an action shall be treated like multiple keyboard hits. This is not part of the classical action recognition task. Where the objective is to know which action is performed. How often it is performed is normally not evaluated.

4.1 System Overview

Figure 4.1 shows a block diagram containing all the necessary computation units to perform a real time action classification further on referred to as *runtime system*. This block diagram shows for simplicities sake a computation chain-based on the assumption that the right scale is known. Therefore avoiding the problem of varying scale. The block digram and the following explanations purpose is to introduce the different computation units of the system and their interaction rather than explaining them in detail which will be done in the upcoming sections.

A camera is capturing images with a certain frame rate. Resulting in a sequence of images denoted as $\mathbf{X}[t]$ where t signifies a time-step and $\mathbf{X}[t]$ represents a signal with a certain sampling rate corresponding

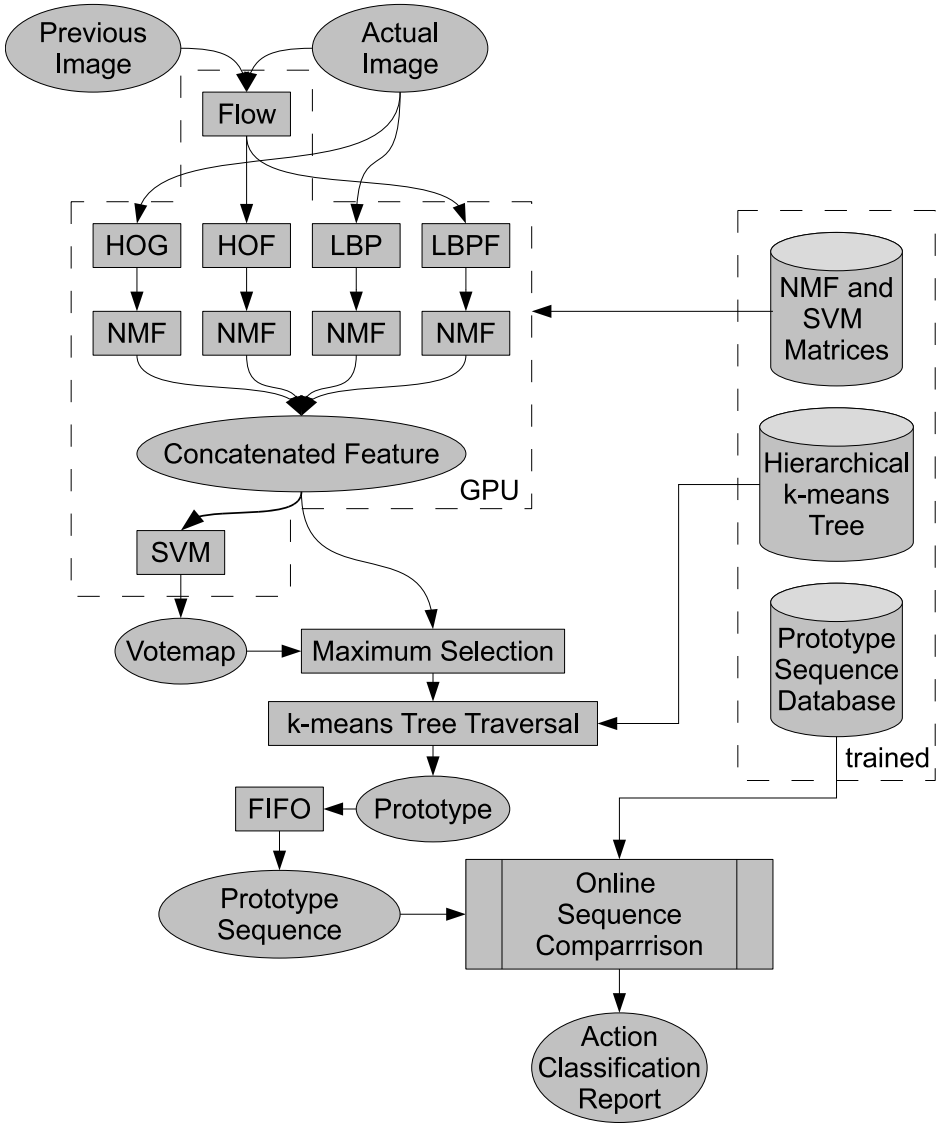


Figure 4.1: A System overview - shows an overview of all the necessary modules involved in the computation. Each box stands for a computation unit while each ellipse stands for an intermediate product of computation. The cylinders indicate stored data. The left part enclosed with the dashed line is computed on the GPU. The right part enclosed with a dashed line is computed in advance by MATLAB and provided to the runtime-system as a file. For a more detailed explanation see the description in section 4.1.

to the frame rate of the camera. At a certain time-step t the two images $\mathbf{X}[t]$ and $\mathbf{X}[t-1]$ are used to calculate the features. These are the *Histogram of Oriented Gradients* (HOG) the *Histogram of Oriented Flow* (HOF) the *Local Binary Patterns* (LBP) and the *Local Binary Patterns of Flow Magnitude* (LBFP). Where HOF and LBFP are depending on flow calculation which is performed by the TV-L1-Flow. The cell sizes of the LBP and LBFP features equals the cell size of the HOG and HOF features.

These features are not only calculated for a single pixel and its neighborhood they are calculated for a patch of the image. Such a path is a two-dimensional region of the image which is of certain defined size, where p_x shall be its predefined width and p_y is its predefined height. p_x and p_y are a not necessarily equal multiples of the cell size. Therefore a patch-feature can be calculated for each patch.

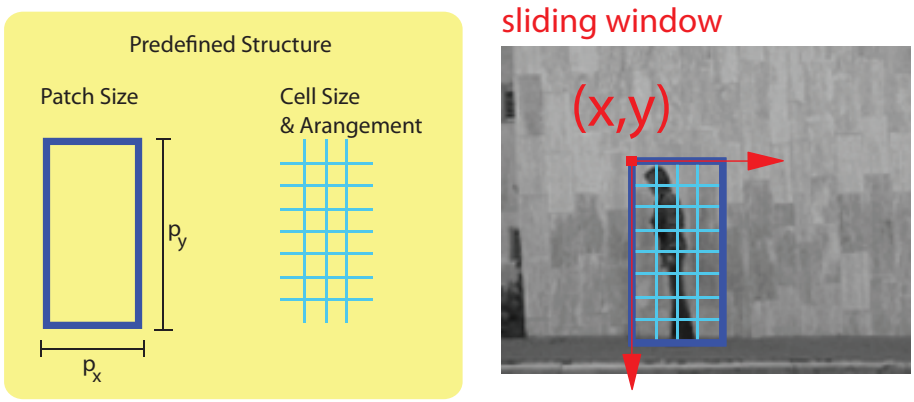


Figure 4.2: Patch-Structure Example for the sliding-window approach - A patch structure of 4×8 cells each of 10×10 pixels is defined. For each (x, y) of the input image a patch-feature is calculated.

In case of HOG and HOF the patch-feature is a concatenation of all the block normalized HOG/HOF-features of all the blocks which fit into the patch (see Sections 3.3 and 3.3.1). While in case of LBP and LBFP they are a concatenation of all the normalized cell-features which fit into the patch (see Sections 3.2 and 3.2.1).

For each of these patch-features a feature dimensionality reduction is performed by a previously determined *Non Negative Matrix Factorization* (NMF) projection matrix. Where these four matrices are different for each of the patch-features. The dimensionality reduced representations of the patch-features are concatenated to a single feature vector per time step t and position (x, y) of the image. Therefore all concatenated patch-feature vectors for a time-step t are a three-dimensional array (as visualized in Figure 4.3). With a position (x, y) the concatenated dimensionality reduced patch-features of the region with the top left corner (x, y) and the bottom right corner $(x + p_x, y + p_y)$ is indexed. This concatenated feature is the input for a linear *Support Vector Machine* (SVM). The linear SVM has been trained to classify the dimensionality reduced patch-features into two classes "part of an action" and "not part of an action". For all pixels corresponding to a feature vector, which was classified as "part of an action" (by $\mathrm{sign}\,()$), the distance to the hyperplane separating this two class problem can be employed as a confidence measure for "being part of an action" (see section 3.5). These confidence measures will be referred to as *votemap* further on.

In contrast to the visualizations in Figure 4.3 and 4.1 the NMF and SVM calculations needed to compute a votemap are actually combined to save memory as will be explained in detail Section 4.3.4. However the functionality stays the same.

Once such votemap has been established its maximum - the pixel with the highest confidence of being the upper left corner of an image region where an action is performed in - is found. The corresponding concatenated feature vector is now analyzed by searching a nearest neighbor in a set of precomputed prototypes. These prototypes have been established by performing hierarchical k-means clustering on all the concatenated features which are of class "part of an action". Therefore the nearest neighbor can be found by traversing the tree in a depth first search pattern.

The resulting prototype is pushed into a *First In First Out* (FIFO) data-structure. By reading the whole

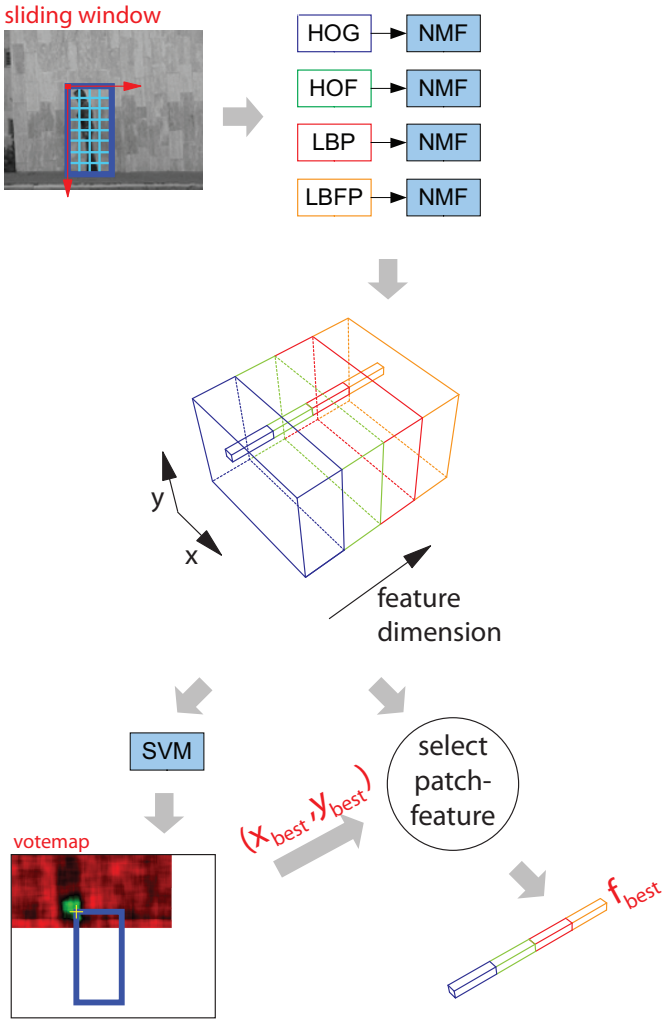


Figure 4.3: Three-Dimensional visualization of the patch features - HOG, HOF, LBP and LBFP are calculated and NMF-transformed for each postion (x, y) of the input image. Therefore resulting in a patch-feature volume of concatenated features. For each (x, y) coordinate a the linear SVM computes the votemap values. The maximum votemap value at $(x_{best}, y_{,best})$ selects the concatenated patch-feature which is assumed to encode the actor information. This patch-feature is used further on.

content of the FIFO buffer a prototype sequence is established. Such a prototype sequence is further on used for the online action classification. Therefore one of two specialized *Dynamic Time Warping* (DTW) algorithms is employed to compare the prototype sequence to previously generated prototype sequences of all the actions to classify. They are the *Adaptive Length Dynamic Time Warping* (AL-DTW) which proved to be not sufficiently robust and computationally to expensive, and a the so called SPRING algorithm.

For both online sequence comparison mechanisms, SPRING and AL-DTW, an action classification report (an action label) is created immediately after this action has been performed. These reports are therefore allowing for repetition detection. In Section 4.8 it will be explained that such reports are related to subsequence matching by DTW and the advantages of SPRING over AL-DTW will be discussed.

4.2 Training

To be able to run such a runtime system several training steps have to be performed in advance. Therefore a dataset of annotated videos is necessary. Such will be discussed in detail in Section 5.1. To be able to run the previously explained runtime-system the NMF projection matrices, the SVM and the k-means-hierarchy have to be trained. Additionally a database of prototype sequences has to be established from the training set. In the following the necessary preparations will be described.

Assume an annotated dataset consists of:

- A set of action-videos of persons performing actions where each video shows a single person performing an action. The action is performed only once and for the whole duration of the video.
- A bounding box enclosing the region of the image where the action is performed for each frame of each video.
- Identifiers for each of these videos representing the labels and target vectors $\{t_1 ... t_N\}$ respectively.
- Additional identifiers for each person.
- A repetition count: There may be several videos of the same person performing the same action. In such cases the repetition counter is different for each of these videos.

Additionally a set of non-action-videos where no action is performed at all is given. It constitutes the negative samples while the action-videos are the positive samples.

The action-videos are prepared in the following way: Each frame is first clipped according to the corresponding bounding box then rescaled to a certain predefined rectangular size, the patch-size $(p_x \times p_y)$. The result is further on called *appearance-patch*. Additionally a flow field is calculated for each pair of consecutive frames of each video. Resulting in a so called flow-stack. This flow-stack can be treated like a video with two channels (dx/dt and dy/dt). For such flow-video the same clipping and rescaling as before is performed. Which leads to the so called *flow-patches*. Since the flow is calculated in pixels the flow magnitude has to be scaled by the same amount as the image is scaled scaled.

This preparations lead to the following per-frame information:

1. appearance-patch
2. flow-patch
3. bounding box
4. action label
5. person identifier
6. repetition count

The negative samples are created by extracting randomly chosen patches from the non-action-videos. Some datasets also provide background videos, showing the same scene as in the action-videos without a performer filmed from the same camera angle. In this case the exact background patches can be extracted and added to the negative samples. For all such negative patches the appearance-patch and the flow-patch can be extracted. All the labels are set to a specific invalid identifier so they can always be identified as negative patches.

For each of the appearance-patches (positives and negatives as well) HOG and LBP patch-features are calculated. For the flow-patches HOF and LBFPs are computed.

The results of the positive samples are used to train the orthogonal NMF matrices. Where the only parameter is the number of base vectors which is also the size of the encoded descriptor. The positive and the negative patch-features are transformed by the corresponding NMF-projection matrix $\mathbf{P}$ (see Section 3.4.3 Equation 3.15). And the results are concatenated. A linear SVM is trained on such concatenated features as a two-class-decision-machine which is classifying into positive and negative samples. Resulting in a decision-surface or hyperplane characterized by $\mathbf{A}$ and b (as described in Section 3.5 Equation 3.29).

The positive patch-features are also used to train a hierarchical k-means. Here the parameters are k and the maximum depth. Then all positive patch-features are traversed through the tree by depth first search until a leaf node is reached which corresponds to a prototype. If all of the patch-features of one video are transformed into frame-prototypes a prototype sequence is established. This sequence is stored in the prototype sequence database to be used by the runtime-system later on (as explained in Section 3.6).

The training is performed in MATLAB . Nevertheless several modules which are necessary for the runtime system and the training as well are actually programmed in C++ and made available to MATLAB as a MEX file. MEX stands for MATLAB executable and allows interfacing between the scripting language MATLAB and the C++ routines. Some of these modules are also incorporating the parallel computing powers of the GPU by using the *Compute Unified Device Architecture* (CUDA) of NVIDIA. Once MATLAB analyzed such a dataset and trained all the necessary entities the information is stored in a file and can be loaded by the runtime system.

4.3 GPU Programming Remarks

Exploiting the massively parallel computation abilities of graphic cards is a topic of interest in the field of computer vision in the last view years. Since the rise of general-purpose computing on graphics processing units (GP-GPU) a programmer is able to program the GPU not only for graphics applications but for various tasks. An algorithm programmed for a CPU is usually run in a single thread, on a single processing unit which has a high clock rate and lots of instructions per second respectively, whereas the same algorithm programmed for a GPU will be divided into lots of threads which run on several processing units in parallel at lower clock rates. Due to this different programing approach fitting design strategies have to be found. These are closely related to the limitations and features of GPU programing. Which is why, after giving a short introduction into GPU programming in general, a basic introduction to the specific features of the CUDA API are given in Section 4.3.1 whereas Section 4.3.2 describes the developed work flow for programming such CUDA enabled programs. Finally Section 4.3.3 derives the necessary design objectives for such programs. The Sections 4.4 and 4.5 describe implementations for specific problems which were created according to such design objectives.

Performance Gain by GP-GPU

If the algorithm and the data which has to be processed fulfill certain requirements, a faster execution on the GPU in comparison to the CPU is possible. The ratio of GPU to CPU execution time is referred to as *speedup*. An algorithm is especially well suited to be computed on a GPU if it can be expressed by data-parallel computations which means the very same program is executed on many data elements in parallel [NVIDIA, 2009b].

In theory for any algorithm exists a upper limit for the performance benefit that can be reached. Therefore code that can not be sufficiently parallelized is better suited in running on the CPU. This upper limit is expressed by Amdahl's law. The maximum speedup S of a section of a program is dependent on its total serial execution time on the CPU P that can be parallelized to be run on N parallel processors of the GPU and can be calculated by

$$S = \frac{1}{(1-P) + \frac{P}{N}} \tag{4.1}$$

Equation 4.1 is only the best possible theoretical outcome. In praxis other factors as memory-bandwidth the GPU-clock and the right usage of caching and pipelining have dominating influence on the actual speedup for a specific algorithm. Such general measures and more detailed directives for programming CUDA can be found in [NVIDIA, 2009a].

Additionally to the performance benefit by parallelizing some usually expensive operations, can be performed very fast on the GPU since they are hardwired in the graphics pipeline. For instance calculating a bilinear interpolated value from an image takes the same amount of time as reading one of its pixels.

4.3.1 CUDA Basics

In this section some CUDA specific basics are introduced since a complete CUDA introduction would exceed the scope of this work. The main purpose is to introduce some of the nomenclature and the according restrictions. The memory used by the CPU or *Host Memory* is different to the one used by the GPU (*Device Memory*). The device memory is faster and its size is limited by the graphic card itself, while the CPU memory is the main memory of the machine and is usually in the range of multiple gigabytes. Data has to be copied between host and device memory. Such transfer operations are slow compared to operations inside each of the memories. The device memory which is accessed by such copy operations is the so called *Global Memory*. Since data-parallel computations are performed on the GPU, the usage of memory is an important factor for programming GPUs. The threads in CUDA are organized in blocks of threads. Within a block the threads are organized in an up to three dimensional arrangement, the *Grid*. The reason for organizing the memory and the threads in such multidimensional structures is a speedup in memory access. Since lots of threads according to the same block are running in parallel they are also reading from and writing to the memory at simultaneously. If the positions of such memory accesses are neighboring in any of the three dimensions the memory access is performed in a so called *coalesced* way. Which means that several memory elements for different threads are physically accessed at once, otherwise the access to memory has to be done sequentially for different threads, meaning they have to wait for each other[1].

A thread itself additionally "owns" some memory, the per-thread *Local Memory* which is very limited and much faster than global device memory but it can not be accessed by other threads.

There also exists the term *Shared Memory* which physically is global memory which is explicitly defined as shared memory. Shared memory is shared for all threads within a block. Accessing shared memory is faster than the access to usual global memory hence it is the preferred method for storing intermediate results. Nevertheless it is also limited in size, dependent on the graphics card.

Additionally a portion of device memory can be assigned to a texture. Which enables the programmer to take advantage of the texture specific features of the GPU. Such are interpolated reading and clamping. However it also enables the so called *Texture Cache*. The reading from textures is performed different than reading from standard device memory. It is cached in a two dimensional way. Meaning that if an element at a certain position is accessed via a texture-read, its surrounding - its 2 dimensional neighborhood - is automatically cached. Once cached the access is as fast as the local memory access. Therefore a texture can be used to avoid uncoalesced reads[2]. This is a read only functionality and the device memory assigned to a texture is not to be altered while used as a texture.

The language used to program the graphics card itself is called "C for CUDA" whereas the surrounding invoking of subroutines which are run on the graphics card is performed in C/C++. The code parts which actually running in parallel on the multiprocessors of the GPU are called *kernels*. CUDA allows every thread to query its position in the arrangement of blocks and cells. From an abstract point of view the invoking of a kernel can bee seen as calling a subroutine, where all the memory management for

[1] There are additional requirements for the alignment of the memory dependent of the generation of graphic card used. Such can be found in detail in [NVIDIA, 2009a] section 3.2.1

[2] In the last generation of NVIDIA graphics cards the global device memory is cached as well. Therefore eliminating the necessity to use textures for caching purposes.

input, output storage on the graphics card has been performed in advance. Like standard C subroutines parameters can be passed to the routine but return arguments are not possible. Additionally the grid and block size are passed to the GPU and they have to be defined by the programmer. The size of shared memory per block is also passed to the GPU.

4.3.2 Testing cycle

The implemented features where integrated into a library developed by the *Institute of Computer Graphics and Machine Vision* (ICG) called *featurelib* which is a library to calculate dense features with CUDA. This library is implying a certain structure for programming, where each feature is not only computable on the GPU but also on the CPU. This redundancy is of advantage for testing purposes but also helps to understand the code better. In the course of this work a MATLAB MEX interface module was added to the library which allowed a generation of MEX files for each of the features. Such MEX files state an interface from MATLAB to the library.

Due to the structure of the task and the library the following work-flow was established:

1. Implement the algorithm in MATLAB as golden standard.
2. Test it mathematically and visualize it if possible.
3. Identify the subproblems which will be implemented as CUDA kernels.
4. Write several C++ routines each performing the task of a single kernel.
5. Compare the whole C++ systems output against the MATLAB version.
6. Implement the kernels in CUDA.
7. Compare their output to the C++ versions output.
8. Analyze the run-times of the features as a whole.
9. Analyze kernel speeds and the number of uncoalesced reads by using the NVIDIA tool *CUDA Visual Profiler*.

Only if all the tests comparisons (point 5 and 7) where correct up to an negligible error an algorithm was considered correct. A performance evaluation is performed in point 8 and 9. Here different implementations where compared in terms of speed. While point 8 evaluates the run-times of the kernels plus the necessary memory management surrounding it, point 9 is only evaluating the kernels but gives very detailed information about their configuration (grids and blocks) and if reads and writes are coalesced. The *CUDA Visual Profiler* also shows a value called *occupancy* which is a percentage of threads which are actually computing in parallel per kernel.

This work-flow is not linear, it is a testing-cycle meaning that changes in the later steps are sometimes requiring changes in the earlier steps.

4.3.3 Objectives for CUDA Design

Unlike a design for a classical CPU program other considerations have to be made for programming CUDA. The following general design guidelines have been established to be able to generate fast and structured code. They are similar to the design of the *featurelib*.

1. Calculate each threads output per pixel of the input image.
2. Use separate kernels for subtasks. Merge them if a significant speedup is possible.
3. Minimize the data which has to be copied between host and device.
4. Avoid uncoalesced reads and writes.
5. Use textures instead of uncoalesced reading.
6. Use shared memory whenever possible but try to keep the data stored in it as small as possible.
7. Keep reused data in the device memory.

Point 7 is going beyond the scope of the structure of the *featurelib* library therefore an unwrapping of the algorithms form the library is performed after an algorithm has been tested within it.

4.3.4 Transformed Features

Objective 3 from above avoids one of the mayor bottle-necks in such system, the CPU-GPU memory transfer. The following example shall give an estimation of the amount of data generated by dense feature calculation:

Assume a patch-feature $\mathbf{f}$ of any kind, with f_d dimensions where each dimension needs f_b bytes to store, which is calculated from a patch of $p_x \times p_y$ pixels for an image $\mathbf{X}$ of $w \times h$ pixels then the final amount of memory needed s in bytes for calculating all patch-features for such an image is given by

$$s = f_d \cdot f_b \cdot (w - p_x + 1) \cdot (h - p_y + 1) \tag{4.2}$$

For example a HOG patch-feature calculated with a cell size of 10×10 pixels, normalized by blocks of 2×2 cells. Where the histogram is quantized in 9 bins. For a patch size of 4×8 cells like used to detect upright full-body performers is resulting in an patch-feature-dimension f_d of 756 float (4 byte) values. Makes $s = 695MB$ for a standard camera resolution of 640×480 pixels. Copying such amount of data makes a GPU-computation pointless since the copying time would exceed any real-time requirement. Therefore a dimensionality reduction is necessary before copying the features back to host memory.

This leads to the following idea: For the identification of the region of interest a votemap has to be created. All necessary computations therefore are performed on the GPU. The votemap is copied to the host memory where the maximum is found and the according patch-feature is calculated on the CPU from intermediate results, which are still stored in the device memory.

If the four features HOG, HOF, LBP and LBFP are analyzed for their effect on the classification result of the support vector machine the following identity can be found. In the following these features are denoted as $\mathbf{f}_{HOG}$, $\mathbf{f}_{HOF}$,$\mathbf{f}_{LBP}$ and $\mathbf{f}_{LBFP}$.

Four different orthogonal NMFs have been trained on patch-features. The result where four projection matrices $\mathbf{P}_{HOG}$, $\mathbf{P}_{HOF}$, $\mathbf{P}_{LBP}$, $\mathbf{P}_{LBFP}$ where each was created from the corresponding NMF-base matrix by $\mathbf{P} = \mathbf{W}^{+}$ as given in Equation 3.15. Additionally a linear SVM has been trained on maximum normalized concatenated NMF-transformed patch-features like explained above, which resulted in $\mathbf{A}$ (the hyperplane coefficients) and its offset to the coordinate system-origin b, then the decision function is given by equation (3.30).

The combination of NMF and SVM can therefore be expressed by

$$y(\mathbf{f}_{HOG}, \mathbf{f}_{HOF}, \mathbf{f}_{LBP}, \mathbf{f}_{LBFP}) = \mathbf{A} \cdot \begin{bmatrix} \mathbf{P}_{HOG} & \mathbf{f}_{HOG} \\ \mathbf{P}_{HOF} & \mathbf{f}_{HOF} \\ \mathbf{P}_{LBP} & \mathbf{f}_{LBP} \\ \mathbf{P}_{LBFP} & \mathbf{f}_{LBFP} \end{bmatrix} - b \tag{4.3}$$

If $\mathbf{A}$ is split column-wise into four parts $\mathbf{A}_{HOG}$, $\mathbf{A}_{HOF}$, $\mathbf{A}_{LBP}$ and $\mathbf{A}_{LBFP}$ where the number of columns of one part fits the number of rows of the corresponding projection matrix and

$$\mathbf{A} = [\mathbf{A}_{HOG}, \mathbf{A}_{HOF}, \mathbf{A}_{LBP}, \mathbf{A}_{LBFP}]$$

equation (4.3) changes to:

$$\begin{aligned} y(\mathbf{f}_{HOG}, \mathbf{f}_{HOF}, \mathbf{f}_{LBP}, \mathbf{f}_{LBFP}) \quad = \quad & \mathbf{A}_{HOG} \quad \mathbf{P}_{HOG} \quad \mathbf{f}_{HOG} \\ & +\mathbf{A}_{HOF} \quad \mathbf{P}_{HOF} \quad \mathbf{f}_{HOF} \\ & +\mathbf{A}_{LBP} \quad \mathbf{P}_{LBP} \quad \mathbf{f}_{LBP} \\ & +\mathbf{A}_{LBFP} \quad \mathbf{P}_{LBFP} \quad \mathbf{f}_{LBFP} \\ & -b \end{aligned} \tag{4.4}$$

where each of the rows are independent from other features. The result of the $\mathbf{APf}$ triples for each feature shall be defined as d. Then $y(\mathbf{f}_{HOG}, \mathbf{f}_{HOF}, \mathbf{f}_{LBP}, \mathbf{f}_{LBFP}) = d_{HOG} + d_{HOF} + d_{LBP} + d_{LBFP} - b$. Each d is the contribution of the corresponding feature to the distance from the SVM-hyperplane.

They can be evaluated on the GPU. Therefore the memory needed to compute all the untransformed features is never copied to the host memory.

4.4 Implementation of HOG and HOF

As already explained in Section 3.3 the calculation of histograms of gradients is very similar to the histograms of flow orientation. The separation of HOG and HOF computation into sub-problems which are implemented as separate kernels is shown in the diagram in Figure 4.4. In the following the HOG computation is explained.

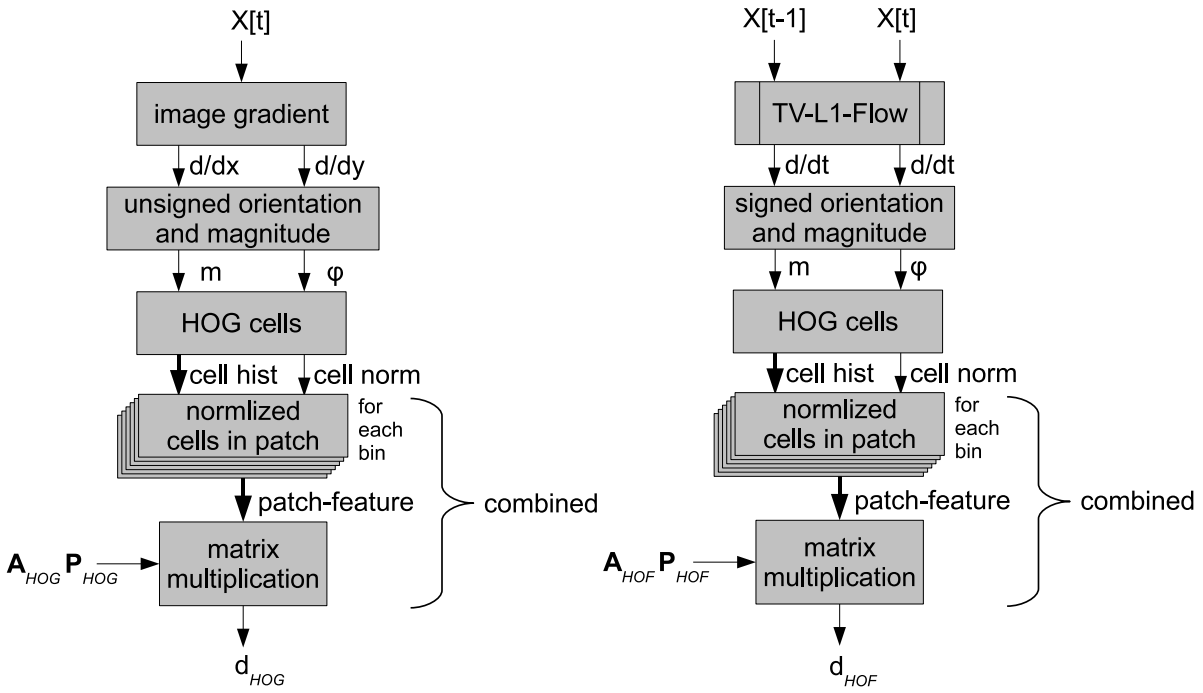


Figure 4.4: Kernels involved in the computation of the HOG and HOF-contribution to the SVM-hyperplane-distance - Shows the kernels for HOG computation on the left and HOF computation on the right. Each of the kernels (simple boxes) acts on a pixel basis. The TV-L1-Flow is not a single kernel but a whole library. The `signed orientation and magnitude` kernel computes a signed orientation in contrast to the unsigned orientation of the HOG. All the thin arrows are two-dimensional in/out-puts while the thick arrows (`cell hist` and `patch−feature`) indicate a three-dimensional array. The hog-block-kernel is invoked several times - for each bin in the cell histograms. The matrix-multiplication-kernel calculates $\mathbf{z}_{out} = \mathbf{A}_{HOG}\mathbf{P}_{HOG}\mathbf{z}_{in}$. Where $\mathbf{z}_{in}$ is the z-dimension of the three-dimensional input. Such is performed for each (x, y) coordinate. Since $\mathbf{A}_{HOG}\mathbf{P}_{HOG}$ is a row vector the output is a two-dimensional array $\mathbf{d}_{HOG}$ which represents the contribution of the HOG feature to the distance to the SVM hyperplane. The last two kernels `normalized cells in patch` and `matrix multiplication` can be combined as explained in Section 4.4.

Orientation and Magnitude

An image $\mathbf{x}[t]$ which is assumed to be a gray-scale image is copied into the device memory. The central difference is computed in x- and y-direction by the `image gradiant` kernel. The result is bound to a texture which is read by a kernel which is computing the unsigned orientation φ and magnitude m of the gradient with the in Section 3.2 defined unsigned orientation definition.

Cell Histograms

The `HOG cells` kernel is using shared memory for the histograms calculation and again a texture is employed for the inputs m and φ. This is both done to avoid uncoalesced memory accesses. The size of the shared memory is given by the number of threads in a block times the `number of bins` of the histogram. Each thread uses only `number of bins` elements of this memory for intermediate results. The thread with the coordinate (x, y) reads all the elements of m and φ which are in the quadratic region

with the top left corner (x, y) and the bottom right corner of ($x+$ `cell size-1`, $y+$ `cell size-1`). It calculates their contributions to the histogram bins (according to figure 3.5) and stores their sum in the shared memory histogram. The squared euclidean norm of the histogram is calculated and the shared memory histogram is copied to the device memory. This copy is done bin by bin and therefore it is a coalesced write.

Normalization and Transformation

To calculate the contribution of the HOG patch-feature to the distance of the hyperplane d_{HOG} two computation chains have been implemented both resulting in the same distances. One needs two kernels and the other only one.

Three tasks are left to be performed by either of this computation chains, they are:

1. Normalizing the cell histograms by the overlapping blocks norms.
2. Building the complete patch-feature.
3. Multiplying this patch-feature with the combined SVM and NMF matrix portion ($\mathbf{A}_{HOG}\mathbf{P}_{HOG}$).

The two-kernel variant performs task 1 and 2 in the first kernel while 3 is performed separately. The one-kernel variant on the other hand performs all of the three tasks at once. It has been developed because of the performance benefit due to less global memory need. In figure 4.4 the two-kernel variant is shown and the kernel fusion is indicated. In the following the two-kernel variant shall be described.

The squared sums of the cell histograms are cached by using a texture. The kernel is invoked once for each bin of the cell histograms to be able to assign the cell histograms to a two-dimensional texture.

The `normalized cells in patch` kernel walks through all overlapping blocks within the patch which is starting with the top left corner at (x, y) of the thread. For each block the block norm is calculated by calculating the square-root of the sum of the former calculated `cell norms` within the block. Then each of the cells inside the block are divided by this norm.

The `matrix multiplication` kernel is performing a matrix multiplication where a one-row-matrix $\mathbf{A}_{HOG}\mathbf{P}_{HOG}$ is accessed via a texture. And a single value per thread is generated.

In the one-kernel variant the patch-feature itself is never stored as a whole only one element of it is calculated in the same way as in the `normalized cells in patch` kernel of the two-kernel variant and immediately multiplied with the corresponding value in $\mathbf{A}_{HOG}\mathbf{P}_{HOG}$. This product is incrementally (for each bin) added to the d_{HOG} which finally sums up to the same result as before. This is the more appropriate variant for the run-time-system despite the fact that it does not allow to reuse the full patch-descriptor.

Histogram of Flow Orientation

The computation of the HOF contribution to the SVM-hyperplane-distance is shown in figure 4.4. It is as already explained in Section 3.3.1 a very similar process to the hog computation. The major difference

is that the input for the `HOG cells` kernel is the signed orientation calculated from dx/dt and dy/dt and the corresponding magnitude m instead of the unsigned orientation computed from d/dx and d/dy.

Comparison to CPU Histogram Computation

As proposed in Section 4.3.2 a CPU variant is also implemented. To compute the histograms by the CPU a integral structure is used. It computes a cumulative histogram. In a cumulative histogram $CumHist(x_{cum}, y_{cum})$ at position (x_{cum}, y_{cum}) the values of the original histograms of all for the positions $\{(x, y) : x < x_{cum}, y < y_{cum})\}$ are integrated. Therefore summing up histogram bins within a cell starting at (x, y) and ending at $(x + c_s - 1, y + c_s - 1)$ where c_s denotes the cell size can be done by

$$CumHist(x + c_s, y + c_s) + CumHist(x, y) - CumHist(x + c_s, y) - CumHist(x, y + c_s)$$

On one hand sing integral histograms is speeding CPU computation because the histograms do not have to be created for each cell separately. On the other hand it involves a dynamic programming approach where at creating the cumulative histogram all the values with smaller x and y have to be known and calculated already. Such dependencies are hard to implement on GPUs. Still, it was tried by using one kennel integrating in x and one integrating in y-direction, but this did not result in the wanted speedup since the data access patterns where not that correlated as in the presented version.

4.5 Implementation of LBP and LBFP

In an initialization phase an LBP-mapping (see Section 3.2) is created by CPU computations. The resulting mapping-table is copied to the device memory to be accessible by the kernels. The kernels access it as texture which gives flexibility in specialization (rotation invariance or uniformity) of the LBP-codes without having to modify kernel code. Since this mapping table is always the same it is copied to the device memory only once.

Cell Histograms

The kernels necessary to compute the LBP-contribution to the SVM-hyperplane are shown in figure 4.5. In the `LBP codes` kernel the computation of the basic LBP-codes and their mapping to a specialized version is performed by texture lookup. The basic LBP-code is computed by sampling points on a circle from the image $\mathbf{X}[t]$. The bilinear interpolation is performed by addressing the image as texture and is therefore much faster than a CPU variant. The `cell histogram` kernel is computing cell histograms of the specialized LBP-codes. These histograms have to store discrete numbers (unlike the HOG and HOF histograms) and therefore a smaller data-type can be used per bin (Only one byte is provided per bin). On the other hand a LBP-histogram has typically much more bins than a HOG-histogram. As in computing the HOG-cell-histograms the histograms are temporarily stored in shared memory. Which turns out to be the limiting factor for the block size.

Therefore the block size is decreased starting from an optimal value (maximal occupation) to a smaller

value to be able to fit the histograms of the block into the shared memory. This computation is dependent on the graphics card used (the amount of maximal shared memory per block), the amount of memory already used and the `number of bins`. After that the `lbp cells` kernel computes the histograms by employing the shared memory in the same way as the `HOG cells` kernel does.

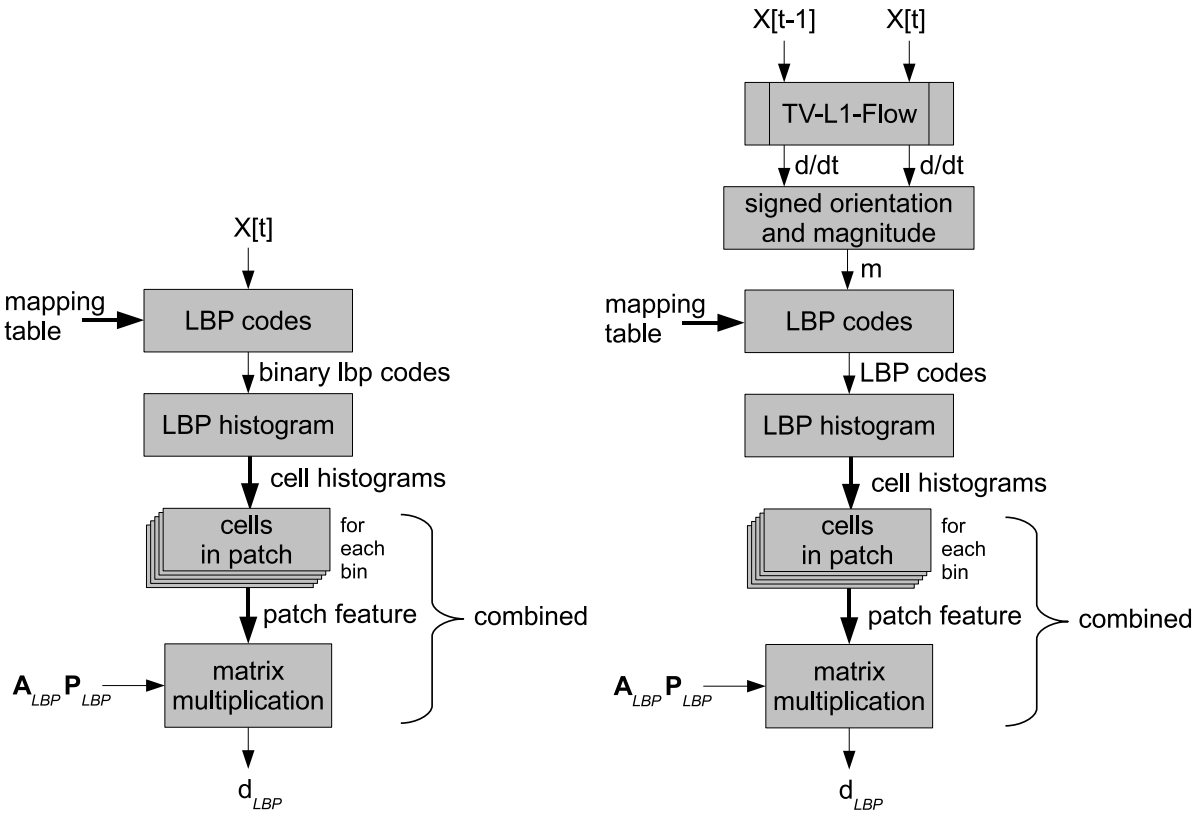


Figure 4.5: Kernels involved in the computation of the LBP and LBFP-contribution to the SVM-hyperplane-distance - Shows the calculation of the distance to the SVM hyperplane of the LBP-feature (left) and the LBFP-feature (right). Each block states a kernel with the exception of the `TV−L1−flow` which is actually a library using several kernels. The thick arrows indicate data transfer of three-dimensional data while thin arrows are two-dimensional (a single value for each pixel position). The `cells in patch` kernel is invoked for each bin sequentially. The last two kernels can be combined as pointed out in 4.4.

Concatenation and Transformation

The `cells in patch` kernel concatenates all the cells within the patch to form a patch descriptor. This is done for each bin separately to be able to employ textures in the same way as in the HOG-kernel `normalized cells in patch`. This can be combined with the `matrix multiplication` kernel like for the HOG-descriptor.

4.6 Actor Detection

The actor detection is performed by a linear Support Vector Machine (SVM) (The according theory can be found in section 3.5). The contributions of all the features to the distance of the SVM-hyperplane are computed on the GPU (See section 4.3.4). Additionally a simple evaluation kernel performs the calculation of $y(x, y)$ according to section 4.3.4 by implementing the formula

$$y(x, y) = \mathbf{d}_{HOG}(x, y) + \mathbf{d}_{HOF}(x, y) + \mathbf{d}_{LBP}(x, y) + \mathbf{d}_{LBFP}(x, y) - b$$

The result are the evaluation results for a patch at the position (x, y) - the so called votemap. As has been shown in Section 3.5 it corresponds to the distance of the SVM-hyperplane. They are copied to the host memory and the maximum distance with a positive sign is treated as the most probable top-left-corner of the patch which encloses the region of the image where the action is performed. This top-left-corner shall be referred to as (x_{best}, y_{best}) further on.

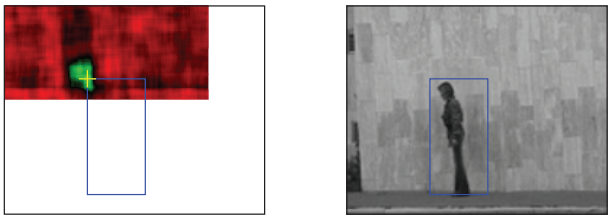


Figure 4.6: Detection of the votemap-maximum and the corresponding patch - Left: The votemap is shown as an image where green indicates positive values and red negative ones. The white region is never calculated since the patch does not fit there. The maximum of the votemap is detected at the yellow cross. Defining the top left corner of the patch (blue rectangle) which states the region of interest. Right: A control image which shows the input image and the detected patch (blue rectangle) which fits the rectangle enclosing the performer.

As one can see in Figure 4.6 the votemap is only calculated for pixels where the patch still fits into the image. To not compute the features for the rim pixels is a design decision. On one hand it avoids strange rim effects and saves memory, but on the other hand it also prevents detections of partially shown actors. This is a disadvantage which is knowingly accepted.

Offset Patches

Since only the feature of the position (x_{best}, y_{best}) is completely evaluated the votemaps correctness and therefore the accuracy of the position (x_{best}, y_{best}) is of great importance for good action-classification results. As in section 4.2 explained the SVM is trained to be able to distinguish the two classes “part of an action” which is the positive class and “not part of an action” which is the negative class. To improve the accuracy of the position (x_{best}, y_{best}) additional negative patches can be extracted and used for training. They are chosen from the positive videos with a significant offset of the original bounding box enclosing the performer (see Figure 4.7). Therefore the local maximum in the votemap is more localized, resulting in a more accurate position.

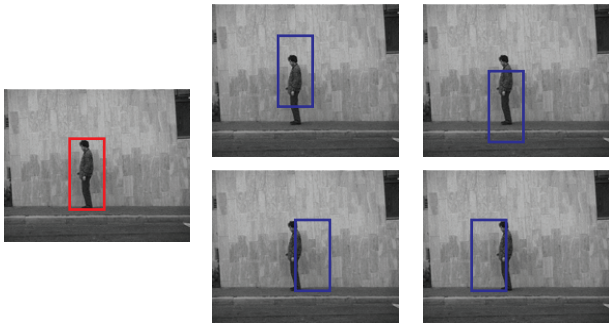


Figure 4.7: Offset patches as negative samples - An example for the selection of offset patches. (left) The original positive annotated patch. (right) The four considered offset-patches. The offset patches are all treated as negative samples.

Determination of Correct Scale

In general the system assumes constant scale during runtime. To ease initialization of the runtime-system it performs a initialization phase where multiple scales are fully computed to determine the best scale to go on with. The user is asked to perform a specific action within the camera view-field the minimal action report for all these scales is treated as the best scale and used for further computations.

4.7 Prototypes Sequences

Once the position (x_{best}, y_{best}) is known, the complete patch-feature has to be generated for this position. Actually the NMF-transformed version of the patch feature is needed. The cell-histograms of all four features are still in the device memory. Only the data which is necessary to compute the patch descriptor at (x_{best}, y_{best}) is copied to the host memory. CPU routines are computing the full patch-feature from this data and are multiplying it with the NMF projection matrices $\mathbf{P}_{HOG}$, $\mathbf{P}_{HOF}$, $\mathbf{P}_{LPB}$ and $\mathbf{P}_{LPBF}$. Once they are concatenated they state the complete dimensionality reduced patch-feature $\mathbf{f}_{best}$.

Once $\mathbf{f}_{best}$ is established, a traversal of the prototype tree is performed resulting in a single prototype per frame. Such prototype tree is visualized in figure 4.8. For visualization purposes this shown tree is generated by NMF-transformed HOG-patch-features only. A prototype for a given $\mathbf{f}_{best}$ corresponds to the leaves of such tree. It is established by k times the tree depth comparisons to the means of clusters. And is therefore fast to be computed. The leaves and prototypes respectively are corresponding to unique discrete identifier values. Such prototypes are the input for a *First In First Out* (FIFO) data structure. A visualization of such a prototype sequence is shown in figure 4.9.

The training of the prototype tree is performed in an hierarchical manner: Starting with all data, the data is clustered by k-means resulting in two clusters since k is chosen to be 2. For each cluster this process is repeated until the variance within a cluster is below a certain threshold or the number of elements inside the cluster is below a minimum cluster size. The algorithm also stops recursion if a maximum depth is reached. Therefore the resulting tree structure is not necessarily very well balanced. Also the number of prototypes is varying for the same data (since k-means is a randomized algorithm).

Experiments have been made where the k-means tree uses the method of data dependent inliers to reject features which are far of the prototypes. Therefore the cluster corresponding to each node is analyzed for their distance to the cluster mean. Only data-points which are closer to the cluster mean than a given percentage of the data-points inside the cluster are treated as inliers. But no enhancement in classification-results could be observed.

The prototypes generated by such hierarchical structure are encoded frame-based representations of the motion and pose of the actor. Only a single value is needed as representation. We will see in the following section that we need to compute distances from prototype to prototype. The distances of the each prototype to all the other prototypes is therefore stored in a distance lookup table. As distance measure for two prototypes the Euclidean distance of the feature vectors of the corresponding cluster-centers are calculated.

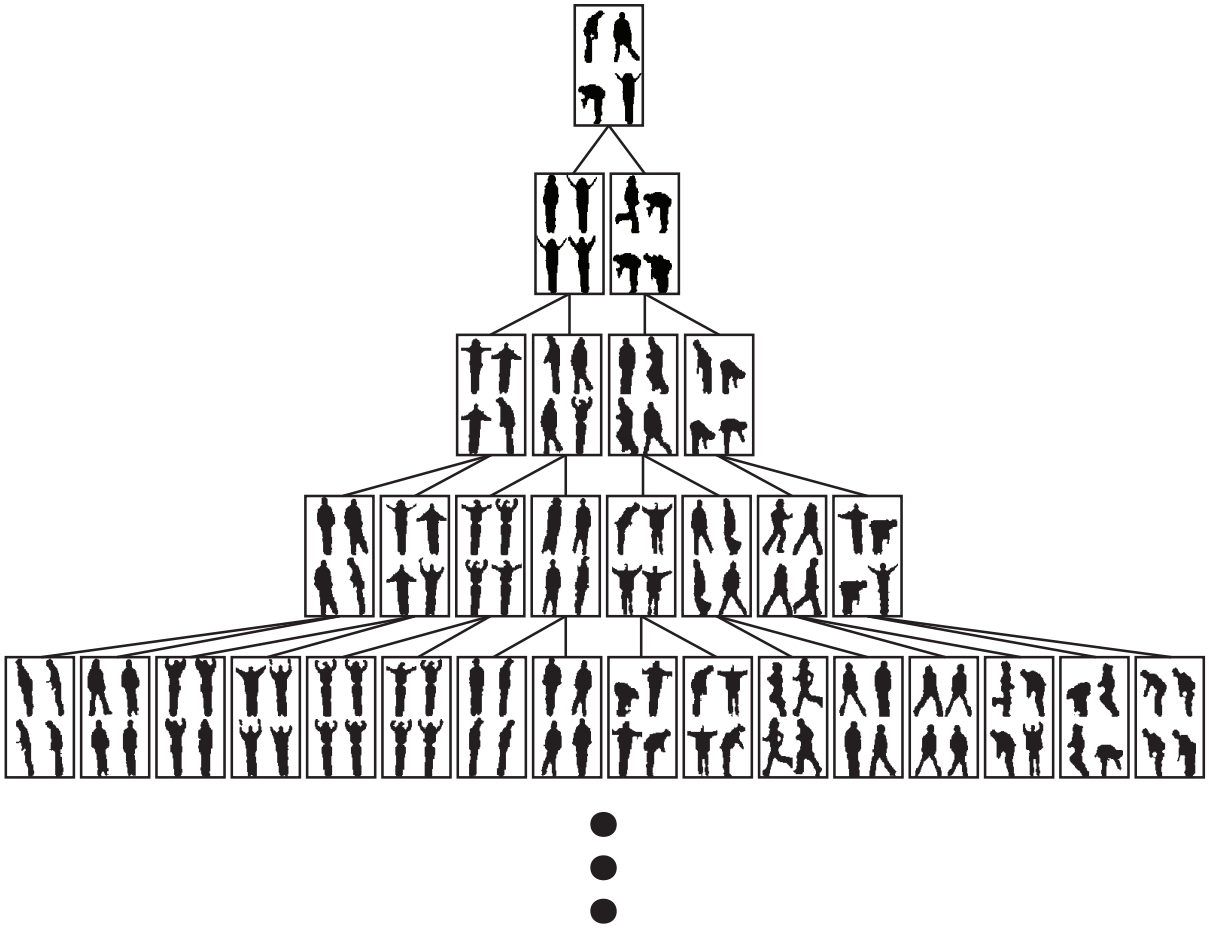


Figure 4.8: Illustration of a hierarchical k-means tree - The image shows the silhouettes corresponding to NMF-transformed HOG-patch-features which have been used to train a hierarchical k-means tree with $k = 2$. Each node is visualized by the silhouettes for randomly chosen samples passing through this node at a depth first search. It can be observed that the left side of the tree tends to consist of upright standing persons where the right part tends to show persons in various other potions. The deeper the tree the more specific a prototype becomes. The tree was trained on HOG-features only since visualizing of appearance is more natural than flow.

4.8 Online Dynamic Time Warping Subsequence-Matching

Once the sequence of prototypes is captured it can be compared to the prototypes within the database. Such is performed by DTW. Since not all of the elements in the captured prototype-stream are part of the action which has to be classified. This is a problem of subsequence matching. The idea is to perform this subsequence matching online, meaning that the process has to be causal.

Two methods will be explained in the following the Adaptive Length Dynamic Time Warping algorithm (AL-DTW) and the SPRING stream specialized DTW method of [Sakurai et al., 2007]. Both are online DTW mechanisms. The major difference is that in AL-DTW the major motive is to emphasize the importance of the end of an action it therefore can be seen as an action-end detector, while SPRING performs classical sub-sequence matching at the earliest possible time-step.

The original DTW definition is using a $dist$ function to calculate the distance between two data points. In our case ,the representation by prototypes (which are corresponding to representatives for a leave cluster), the prototype distance lookup-table calculated during training is used to compute the distance, which significantly speeds up the process (see [Lin et al., 2009]).

4.8.1 Adaptive Length Dynamic Time Warping

As explained in section 3.7 to compute the cost of a dynamic time warping path a distance matrix has to be generated by a dynamic programing scheme and the last element of it $\mathbf{D}(N, M)$ is the DTW-cost of the whole path. To compute only the cost the path itself must not be known.

In [Lin et al., 2009] a normalization of this DTW cost by the path length was proposed for the purpose of action-classification. Therefore the path length has to be known. This is possible by extracting the DTW path by reversed traversing of $\mathbf{D}$ as explained in section 3.7.

Another possibility is to create a length matrix $\mathbf{L}$ simultaneously with the generation of the distance matrix $\mathbf{D}$. At each dynamic-programming-step during the calculation of $\mathbf{D}$ the cost of all shorter warp-paths are known, as shown in equation (3.34). A length matrix $\mathbf{L}$ can therefore be computed dynamically by

$$\begin{aligned} \mathbf{L}(\mathbf{x}) &= 1 + \mathbf{L}(\mathbf{x} + \underset{d \in \mathbf{Neigh}}{\operatorname{argmin}}(D(\mathbf{x}+d))) \\ \text{with} \quad \mathbf{Neigh} &= \{(-1,0)^T, (0,-1)^T, (-1,-1)^T\} \end{aligned} \tag{4.5}$$

with $\mathbf{x} = (n, m)^T$ as indexing vector for the distance matrix $\mathbf{D}(\mathbf{x}) = \mathbf{D}(n, m)$ and $\mathbf{Neigh}$ representing the neighboring positions in $\mathbf{D}$.

The advantage of computing $\mathbf{L}$ and $\mathbf{D}$ simultaneous is that the normalized costs (by the length of the path) can also be computed immediately. For each element of $\mathbf{L}$ and $\mathbf{D}$ the normalized cost $Dist_{norm}(n, m)$ is computed by

$$Dist_{norm}(n, m) = \mathbf{D}(n, m)/\mathbf{L}(n, m).$$

and the AL-DTW algorithm finds the minimum of these costs by

$$cost = \min_{\substack{n_{min} \leq n \leq N, \\ m_{min} \leq m \leq M}} Dist_{norm}(n,m) = \min_{\substack{n_{min} \leq n \leq N, \\ m_{min} \leq m \leq M}} \frac{\mathbf{D}(n,m)}{\mathbf{L}(n,m)} \tag{4.6}$$

where n_{min} and m_{min} are the minimum number of elements in $\mathbf{X}$ and $\mathbf{Y}$ which have to be used to construct the warp path. n_{min} and m_{min} are parameters to the algorithm. As additional output of the algorithm the m and n which where chosen in equation (4.6) are returned. The complete signature is therefore

$$\text{AL-DTW}(\mathbf{X}, \mathbf{Y}, n_{min}, m_{min}) \rightarrow [cost, n_{opt}, m_{opt}]$$

The minimization problem of Equation (4.6) is solved at the same time as the DTW matrix $\mathbf{D}$ is calculated. The algorithm has the same complexity as the standard DTW algorithm in terms of space as well as memory ($\mathbf{O}(N * M)$) but is optimizing for the additional minma in Equation (4.6). The naive approach of running the standard dynamic time warping algorithm multiple times for each n and m would need $\mathbf{O}((N * M)^2)$.

If this normalized-cost minimum is used for the comparison of the two time-series, divergence at their end is allowed without penalty.

Action Classification

The action classification is performed by comparing the contents of the prototype-sequence-database with the current content of the FIFO-buffer. The objective here is to compute a distance measure which is emphasizing the importance of the end of an action.

We are dealing with a causal system where the data in the prototype-FIFO-buffer is only dependent on observations in the past. At a time-step t only the past prototypes of time-step $t - W$ to time-step t are accessible. But not all of this prototypes are actually part of the action. Therefore certain flexibility in the number of consecutive past prototypes which is actually used for cost calculation is desired.

The contents of the database can be seen as candidates $\mathbf{c}_i$ of variable length, whereas the content of the FIFO buffer represents a query sequence $\mathbf{q}$ of length W. Each entry of a certain $\mathbf{c}_i$ or $\mathbf{q}$ has only one dimension which is a numerical identifier for the prototype. A precomputed distance matrix $\mathbf{D}$ containing the euclidean distance of each prototype to each prototype is generated as a distance lookup table.

As the AL-DTW algorithm allows divergence at the end of a time series without penalty, the query $\mathbf{q}$ and the candidates $\mathbf{c}_i$ have to be reversed in time before provided to the algorithm to focus on their end.

$$\text{AL-DTW}(flip(\mathbf{q}), flip(\mathbf{c}_i), q_{min}, c_{min}, \mathbf{D}) \rightarrow [cost, n_{opt}, m_{opt}]$$

If done so the effect is that the AL-DTW algorithm is allowing for divergence of the query and the candidate at their beginning. By introducing the lookup-table for distances $\mathbf{D}$ the implemented is faster. Additionally a Sakoe-Chuba band boundary is implemented for additional speedup.

The amount of optimal and obligatory time-steps for the sliding window is also dependent on the candidate (not indicated in figure 4.9). Since for a short candidate a smaller number of obligatory time-steps in

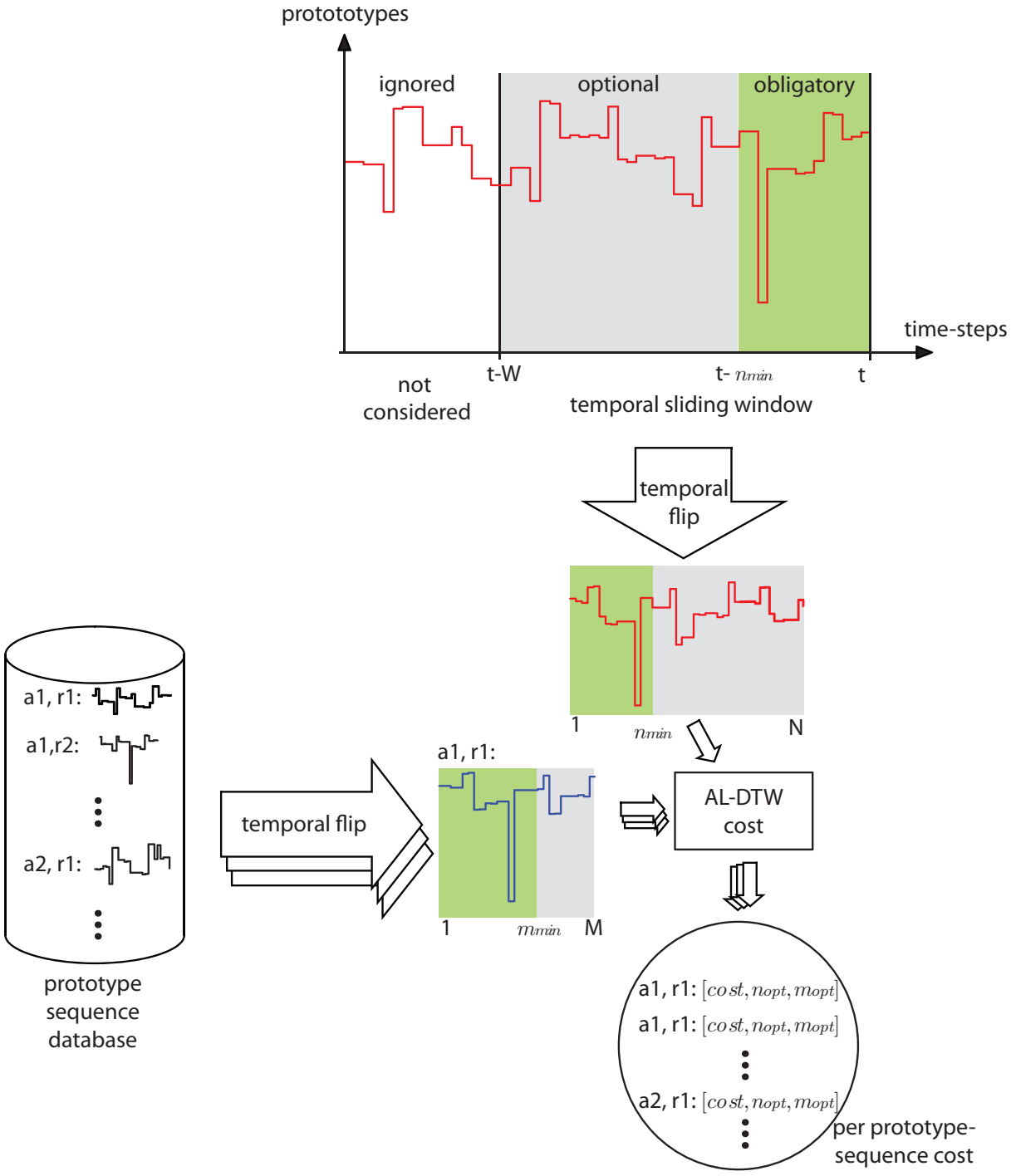


Figure 4.9: Prototype Analysis - A prototype sequence read from the FIFO buffer (the temporal sliding window) has an obligatory and an optional part as it is not known how long the sequence which fits best will be. Its length is $t - W + 1$ if W denotes the length of the temporal sliding window. A temporal flip is performed to the sequence before it is compared to the temporally flipped candidates in the prototype sequence database by *Adaptive Length Dynamic Time Warping*. The candidates in the database are annotated with the corresponding action label (target vector) and the repetition number. The result of all of this comparisons is a cost, a best fitting query length n_{opt} and a best fitting candidate length m_{opt} per candidate.

$\mathbf{q}$ is needed than for a long one. Therefore it is possible to use the system for action detection of actions which have significantly different durations.

Action End Detection

As previously pointed out, the AL-DTW-cost is minimal if the FIFO buffer contains a whole action and its end is perfectly reached. Therefore a temporal minimum in AL-DTW-costs can be employed to detect such ends.

The problem is that due to single mismatches in the prototype-sequences, a local minimum in AL-DTW-costs occurs multiple times per repetition of an action. Figure 4.10 shows such action end detections. Two strategies have been developed to overcome the problem. The first is simply convolving the time series of AL-DTW-cost with a small Gaussian convolution kernel of size G. Such would lead to a "reasonable local minimum" meaning that getting stuck in a tiny local minimum is avoided.

The disadvantage is, that for such calculation $G/2$ values in the future would be needed, resulting in acausal system. Such is not possible for the run-time system. In a run-time system the only possibility for flattening a time series by a Gaussian kernel is to convolve the values inside the temporal window of size W with a Gaussian kernel of size G. This is resulting in a buffer with $W - \lceil G/2 \rceil$ elements. Therefore $\lceil G/2 \rceil$ is the minimum delay which can be reached. If a camera refresh rate of 15 frames per second is assumed a Gaussian kernel, which is performing well (of size 31), would already mean $\lceil G/2 \rceil / 15 = 1.06$ seconds of latency. This is to much for a real time system which shall act as human-computer-interface.

Therefore another approach is taken. The idea is to use the knowledge of how long an action is to improve the minimum detection. Therefore the constraint that the trigger is accruing at the optimal position where the minimum of costs is reached at a "reasonable local minimum" is released. It shall be allowed that the trigger is generated earlier than the trigger generated by a "reasonable minimum" detection would be. A minimum is detected if the backward difference is negative or zero and the forward difference positive. Which can formally be written as

$$\text{IS-MIN}(x[t]) := (x[t] - x[t-1] \leq 0) \wedge (x[t+1] - x[t] > 0)$$

To make this equation causal we can shift the temporal index by one $x[t+1] \rightarrow x[t]$, resulting in

$$\text{WAS-MIN}(x[t]) := (x[t-1] - x[t-2] \leq 0) \wedge (x[t] - x[t-1] > 0)$$

and a latency of a single frame. Such alone would result in by far to many detected minima, but an additional suppressing of minima dependent on the action which is detected at such minimum leads to fairly stable results.

The suppression is based on the following idea: The best fitting query length n_{opt} of the AL-DTW algorithm determines the supposed position of the beginning of an action. Therefore we only allow a minimum do be detected if within the temporal window from $t - n_{opt}$ to t no minimum was detected.

Alternatively the same is tried with m_{opt} (the candidate length which is employed for the cost calculation) and the candidate length itself (M). But n_{opt} outperforms the other variants.

Figure 4.10 compares the two minima detection methods.

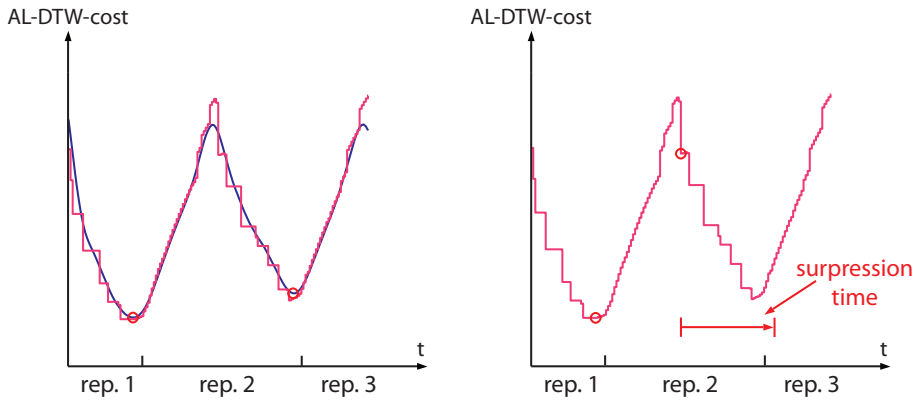


Figure 4.10: Comparrisson of two local temporal minma detection methods for the AL-DTW-cost - (left) Minima detected by Gauss-convolution and appliance of WAS-MIN. The smoothed step-function (blue) is computed by convolution with a Gauss-kernel of size 15. (right) Minmas detected by WAS-MIN and suppression of minima for the duration of n_{opt}. The step-function is indicating the AL-DTW-costs for given video. In the shown time period three repetitions of the same action have appeared. One of these is not finished jet. A significant reduction of costs can be observed at the end of an action. The number of minima detected is the same in both variants whereas the position (the time-step t) where the minimum is detected varies.

Additionally to being a local minimum a local cost-minimum has to be lower than a given threshold to be valid. This threshold is determined during training by cross evaluation.

After testing this algorithm, it showed to be working to a certain amount. But it also turned out be very sensitive for noise. False detections of repetitions occurred very often and the determination of the parameters for the filtering steps turned out to be very hard, as will be shown in detail in the evaluation section (Section 5.2.1). Due to its significant computation time it also causes a noticeable drop in frame rate, which inherently makes the system more error-prone, since informations where lost by temporal under-sampling. Therefore an alternative is searched and the SPRING algorithm (explained in the following section) which outperforms AL-DTW by speed as well as accuracy is found and implemented.

4.8.2 Stream monitoring under Dynamic Time Warping

In a live action recognition system one prototype is the result of one camera frame. Therefore the sequence of prototypes acquired in the past is extended by one prototype with each time step. The SPRING algorithm introduced by [Sakurai et al., 2007] is addressing the problem of subsequence matching for such streaming data, while minimizing the number of computations and the amount of memory needed for each time-step by reusing the accumulated distance matrix $\mathbf{D}$ acquired up to a certain time step for all upcoming time-steps (see Figure 4.11).

The ideas behind the algorithm are:

1. By padding the candidate sequences $\mathbf{Y}$ containing m elements, with a symbol which evaluates to zero distance only $O(m)$ numbers have to be updated per time step (proven in [Sakurai et al., 2007]).

2. To be able to determine the starting-time of a warping path in such streaming fashion an additional matrix $\mathbf{S}$ is computed which records the starting-times of the warping paths. The element $\mathbf{S}(n, m)$ contains the starting-position for the warping path ending at $\mathbf{D}(n, m)$.

3. Additionally to $\mathbf{S}$ and $\mathbf{D}$ and the original SPRING definition of [Sakurai et al., 2007] a matrix containing the path length is introduced. An element of $\mathbf{L}$ contains the length of the warping path ending at the corresponding element in $\mathbf{D}$. Therefore the length of the warping path is known for each element of $\mathbf{D}$ and the costs can be normalized by the warping path length, like proposed in [Lin et al., 2009].

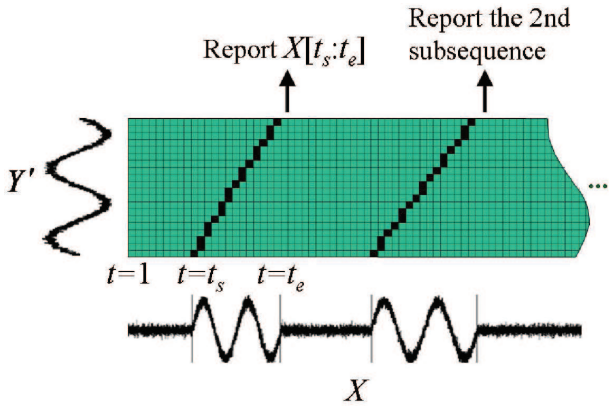


Figure 4.11: The SPRING procedures datastructure - The augmented candidate-sequence $\mathbf{Y}$' is searched for in the data stream $\mathbf{X}$. For each subsequence-match a report is generated as early as possible (copied from [Sakurai et al., 2007])

.

If we assume $\mathbf{X}$ to be the stream and $\mathbf{Y}$ to be the sequence searched for, than $\mathbf{Y}'$ shall be generated by adding a specific symbol ("*") to its beginning. Sakurai refers to it as star padding:

$$\mathbf{Y}' = [*, \mathbf{Y}]$$

Whenever a distance to this symbol is calculated it evaluates to zero.

Actually, in contrast to the visualization in Figure 4.11, there are only two columns of $\mathbf{S}$, $\mathbf{D}$ and $\mathbf{L}$ required for the computation, the current and the previous value. The current values shall be denoted as $\mathbf{s}$, $\mathbf{d}$ and $\mathbf{l}$, while $\mathbf{s}'$, $\mathbf{d}'$ and $\mathbf{l}'$ are the past values.

The i^{th} element in the distance column at timestep t can be propagated by

$$\begin{aligned} d_i &= dist(X[t], Y[i]) + d_{best} \\ d_{best} &= \min\left\{d_{i-1}, d'_i, d'_{i-1}\right\} \\ d_0 = d'_0 &= 0 \end{aligned} \tag{4.7}$$

This propagation performs the same operation as the classical DTW propagation-scheme of Equation 3.34 by only using two columns. If the next timestep is reached $\mathbf{d}$ is stored in $\mathbf{d}'$ and the new $\mathbf{d}$ can be

calculated the same way. In a similar manner s and l is propagated

$$s_i = \begin{cases} s_{i-1} & \text{if } d_{i-1} = d_{best} \\ s'_i & \text{if } d'_i = d_{best} \\ s'_{i-1} & \text{if } d'_{i-1} = d_{best} \end{cases} \tag{4.8}$$

$$l_i = \begin{cases} l_{i-1} + 1 & \text{if } d_{i-1} = d_{best} \\ l'_i + 1 & \text{if } d'_i = d_{best} \\ l'_{i-1} + 1 & \text{if } d'_{i-1} = d_{best} \end{cases} \tag{4.9}$$

Input: A single value of the stream $\mathbf{X}$ at timestep t: $x_t = \mathbf{X}[t]$ and a subsequence to search for (which stays constant over time ($\mathbf{Y}$ with m elements).

Output: A subsequence-match consisting of $(cost, t_s, t_e)$ the $cost$ equals the minimal normalized (by length) DTW distance. t_s is the starting time-step while t_e denotes the end of the detected subsequence-match.

1: **for** $i = 1$ to m **do**
2: Compute $\mathbf{d}$ and $\mathbf{s}$ and $\mathbf{l}$ according to Equations (4.7), (4.8) and (4.9).
3: **end for**
4: **if** $d_{norm,min} \leq \epsilon$ **then**
5: **if** $\forall_i \{d_i/l_i > d_{norm,min}$ or $s_i > t_e\}$ **then**
6: Report $(d_{norm,min}, t_s, t_e)$
7: $d_{norm,min} = \infty$
8: **for** $i = 1$ to m **do**
9: **if** $s_i \leq t_e$ **then**
10: $d_i = \infty$
11: **end if**
12: **end for**
13: **end if**
14: **end if**
15: **if** $d_m \leq \epsilon$ and $d_m < d_{norm,min}$ **then**
16: $d_{norm,min} = d_m/l_m$
17: $t_s = s_m$
18: $t_s = s_m$
19: $t_e = t$
20: **end if**
21: $\mathbf{d}_i \rightarrow \mathbf{d}'_i$
22: $\mathbf{s}_i \rightarrow \mathbf{s}'_i$
23: $\mathbf{l}_i \rightarrow \mathbf{l}'_i$

Figure 4.12: The SPRING Algorithm - Takes one value of the stream at a time. And generates a report if a subsequence-match is found which evaluates to a normalized distance smaller than ϵ and no cheaper normalized distance can be accomplished at later time-steps. (For a detailed description see the text.)

The algorithm is shown in Figure 4.12. During the evaluation SPRING is always remembering the best warping path which has a normalized cost less than ϵ (denoted as $d_{min,norm}$) and the according start and end time 15 to 20. And is reporting a repetition if no cheaper path can be accomplished. This condition is fulfilled if all current normalized distances are bigger than the minimal normalized distance or all current

starting times are greater than the end time of the subsequence match (as shown in Line 5 of Figure 4.12). After such report has been performed it is of importance to reset the structures as shown in Line 7 to 12.

A report consists of start and an end-time plus the according normalized time-warping-distance. The value of ϵ is determined during training by cross-evaluation. In contrast to the original SPRING implementation, to gain less latency, it is also reporting a path if a certain number of time-steps since its occurrence have passed (not shown in the Algorithm).

Classification by SPRING

Since multiple candidate-sequences are compared to a single query sequence an additional report-filter-step is introduced (see Figure 4.13). It works as follows: If multiple reports of different candidate-sequences occur simultaneously only the minimal cost report is used. The system assumes all reports which have occurred up to the current time step, to be correct. It only allows reports which have a starting-time which is greater than the last end time, but it allows a temporal overlap up to 25%.

In contrast to AL-DTW the whole sequence of $\mathbf{Y}$ is incorporated in the result. And the detection of a unambiguous starting-time can be used for more sophisticated filtering of reports.

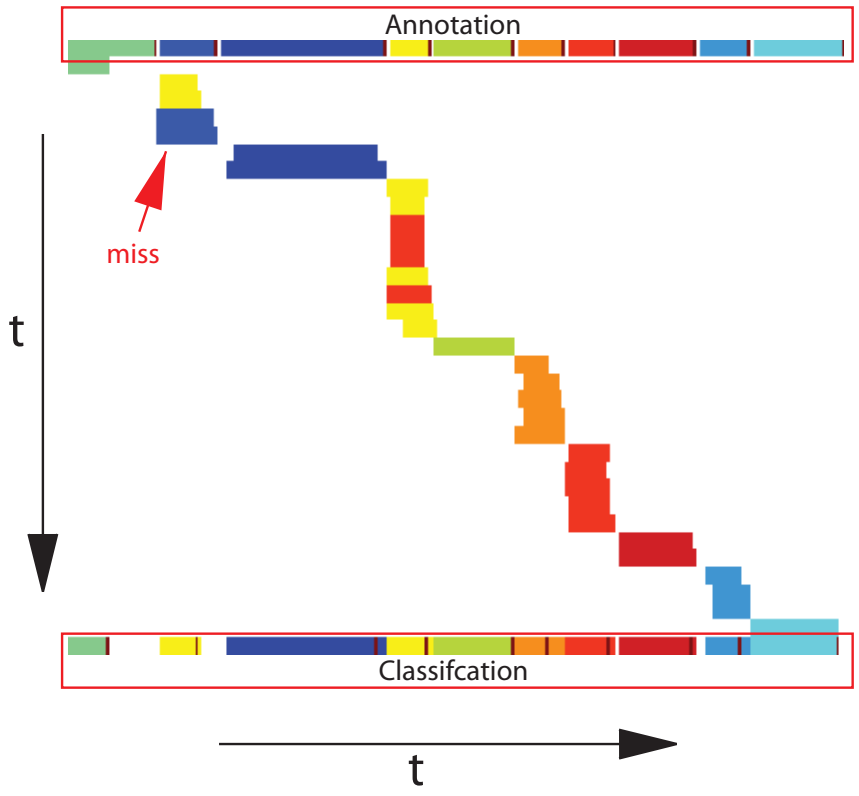


Figure 4.13: The action classification reports versus their occurrence over time - Each color represents an action. The temporal alignment of different actions is shown horizontally. The graphic has one line for each occurrence of a report. Due to the additional filtering dealing with concurring reports the yellow report (at the miss-indicator) is the classification result since it occurs first even if the blue one would be the right classification.

Chapter 5

Experiments

To evaluate the functionality of the proposed system and to be able to compare it to other published works evaluation is performed on various datasets. The datasets and their preparation is discussed in Section 5.1 while the results are presented in Section 5.2. Additionally the video streams where generated which emulate a sequence of repetitions of various actions. Their results are presented in Section 5.3.

5.1 Datasets and their Preparation

To evaluate the system three human body action recognition databases are used. These are the Weizmann dataset [Gorelick et al., 2005], the KTH dataset [Schuldt et al., 2004] and Keck-Gesture dataset [Lin et al., 2009].

Weizmann Dataset

The Weizmann dataset consists of a number of videos with nine actors performing ten actions as can be seen in Figure 5.1. The number of times an action is performed is varying from one to ten times. The annotation does not contain start or end times of a repetition nor a clear definition what constitutes a repetition. To be able to validate on this dataset this start and end time annotations are added manually. After watching the videos and analyzing their content definitions for "what constitutes one repetition of each action" are defined. An example repetition definition for the action bend: "An upright standing person in lateral position bends down like he/she would pick up a coin and gets up again until he/she is standing in an upright position again." These repetition-definitions are illustrated in Figure 5.1. Additionally the desired patch size is defined as 40×80 pixels. The original annotation of the Weizmann dataset consist of:

- Binary masks, which are the result of foreground-background subtraction of the videos
- The action label - which action is performed
- The name of the person performing it

To be compatible with the defined training structure (as explained in Section 4.2) the following preparation is performed: For each frame a bounding-box enclosing all true values in the according binary-mask is generated using Matlab. During evaluation scale stays constant over consecutive frames, therefore the scale selection during training has to be constant as well. The maximum height h_{max} of all bounding-boxes of a repetition is used to determine the correct scale for the whole repetition. The bottom extrema of the mask are most likely to be the feet. Therefore the anchor position of the final bounding box is determined by averaging the two bottom extrema of the mask. Once scale and anchor position are known the video can be scaled by $80/h_{max}$ and the patch can be extracted. The same scaling and cropping is used for the flow, which has been computed in advance. Since flow is measured in the unit of pixels which is scale dependent its magnitude has to be scaled as well.

Additionally to the original patches extracted for training the set was extended by adding a vertically flipped version of all patches. The reason is, that for example walking from the left to the right side would not be detected if trained only with walkers who walked from right to the left side.

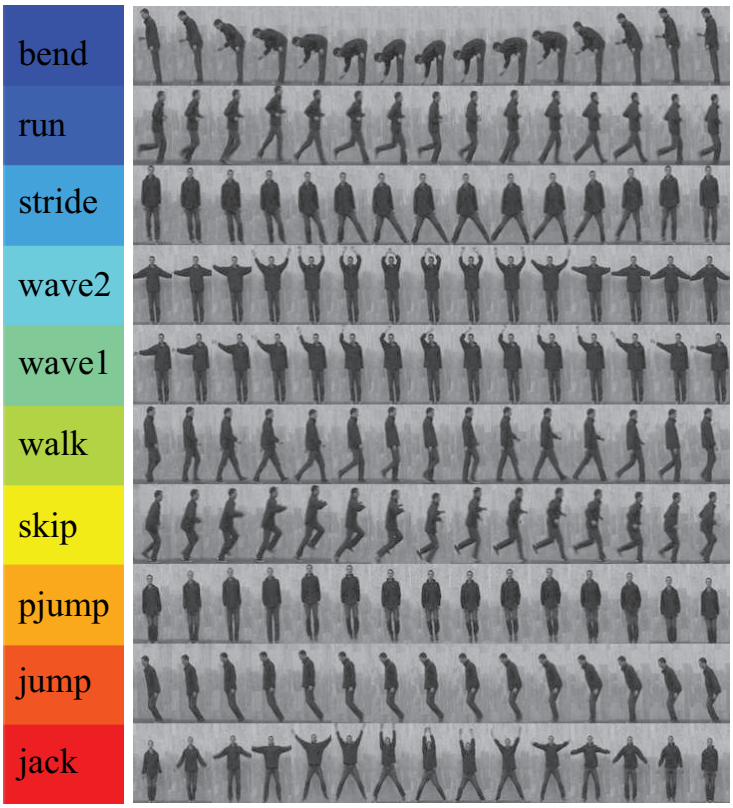


Figure 5.1: Repetition annotation for the Weizmann Action Dataset - The ten actions version of the Weizmann Dataset is used. A patch consists of 4 $\times$ 8 cells of 10 $\times$ 10 pixels. Each line of images represents a single repetition of an action where the defined beginning is the first image and the defined end the last one. For visualization purposes only, the frames in-between have been sampled uniformly in time.

For evaluation a training and an evaluation-set have to defined. The method of cross evaluation is used. The evaluation-set are all videos of one specific performer, while all other videos state the training-set. Such partitioning, into the pair of training and evaluation-set, is performed nine times (for each person) and the evaluations are performed for each of these pairs. The final performance result is the average

results of these evaluations (such is referred to as “leave one person out” technique).

KTH Dataset

The KTH dataset consists of six actions performed by 25 persons in four scenarios. However only the first scenario is used in this work. All of these actions are visualized in Figure 5.2. There is also shown how the patches where selected and how the actions where defined. The definitions of the repetitions is much harder in this case. The boxing action for example is varying a lot, some actors behave like they are actually in a fight, meaning they are boxing multiple times and very fast with one hand than they pretend to be blocking before they are dodging. Such actions are problematic for the given approach of analyzing a defined sequence of poses as can be seen in the results presented in Section 5.2. The KTH Dataset itself does not contain any reasonable bounding-box or foreground-background based actor-location information. The only location-information available was manually generated (by the ICG institute) but only for a small number of frames per video. The problem is that for some actions no complete repetition is to be found in these annotations. An already existing Matlab location-annotation tool is modified to be able to generate new annotations which are consisting of repetitions and locations for each action. The mentioned problems are the reason why only the S1 subset of the KTH-dataset is used for evaluation. In contrast to the Weizmann dataset the camera is moving. To compensate the influence of the camera movement the mean optical flow is subtracted. Similar to the Weizmann dataset the division into evaluation and trainings-set is performed by the leave one person out cross evaluation technique. The patch size and cell arrangement $\times 8$ cells of 10×10 pixels was chosen for patch extraction. And all patch extraction was performed like in the Weizmann dataset. Flipped patches where extracted for the same reason as above.

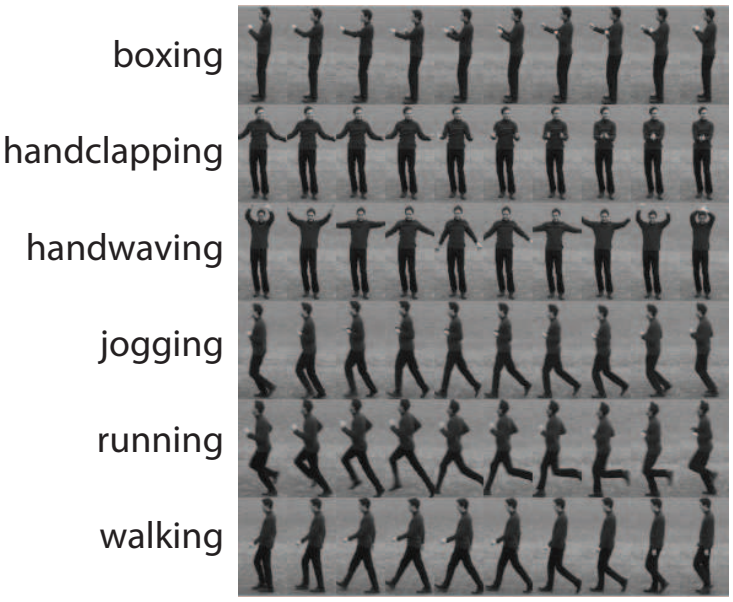


Figure 5.2: Repetition annotation for the KTH Actions dataset - A patch consists of 4×8 cells of 10×10 pixels. Each line of images represents a single repetition of an action where the defined beginning is the first image and the defined end the last one. For visualization purposes only, the frames in-between have been sampled uniformly in time.

KECK Dataset

The KECK gesture dataset consists of a training set of 14 different gestures which are military signals each performed by three different persons, each repeating them three times. In the training setup the scene is very simple, since the actors are performing in front of a green sheet. As location-information binary foreground-background subtraction masks are provided. The test-set contains problems as moving camera and other persons passing through the scene as noise and plays in different scenarios. For some videos the background contains human-like structures in the background. The annotation of repetition-sequences is straight forward since gestures have an obvious cycle. The bounding-box for the patches has been selected like visualized in Figure 5.4 and the resulting patches are shown in Figure 5.3. Since the camera is moved the mean image flow is subtracted before flow patches are extracted like with the KTH dataset.

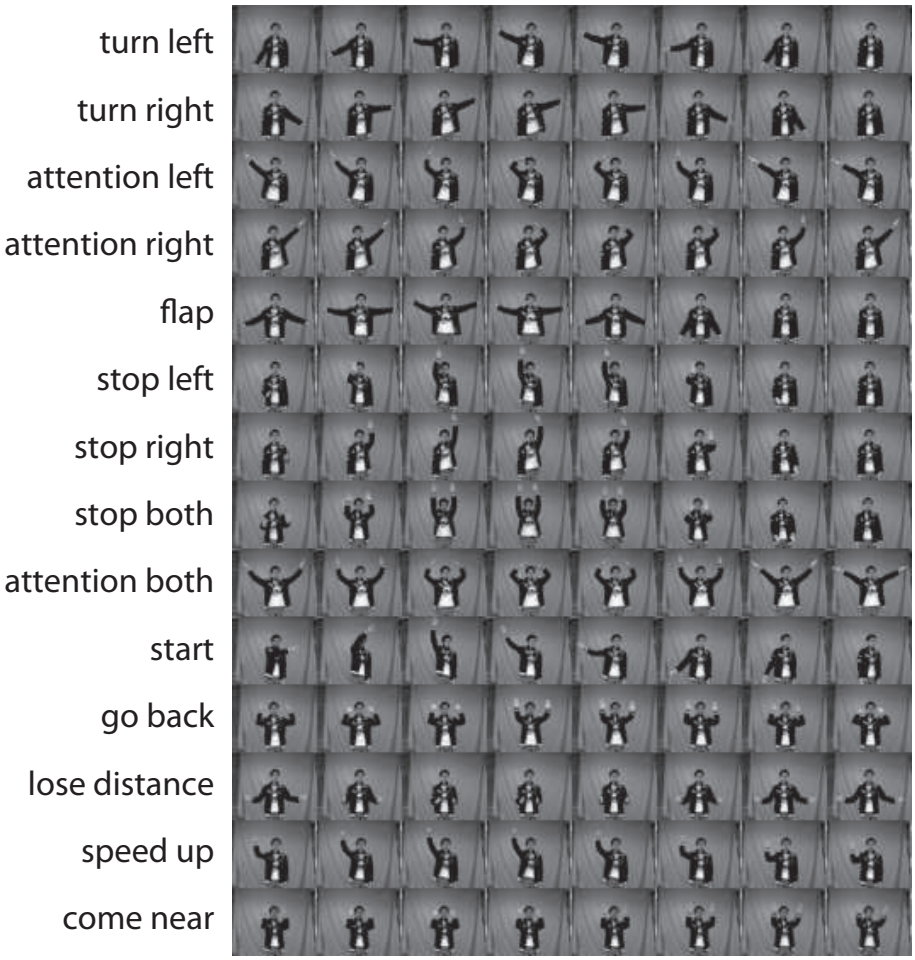


Figure 5.3: Repetition annotation for the KECK Gesture dataset - A patch consists of 4×8 cells of 10×10 pixels. Each line of images represents a single repetition of an action where the defined beginning is the first image and the defined end the last one. For visualization purposes only, the frames in-between have been sampled uniformly in time.

```
% keck_determine_patches extracts the following:
%  B ... Foot point the mean of the bottom extremas
%  C ... Centere in respect of the arms movements (colinear with B in
%        y-direction )
%        is also the center of an rectangle containing every possible arm
%        movement this rectangle is the final bounding box.
%  All images of an action performed by one person get scaled by the same
%  amount. Which is determinded by the minimal height a person has during
%  performing an action
%  ( should be the distance of the head position to B ).
%  Then following relative proportions are assumed:
%
%                        _                                  _
%                        |                                  |
%                        |                                  |
%                        |                                  |
%           .-.          _                 _                |  0.5 (arm length)
%          (o.o)         |    0.79         |                |
%           (-)          |                 |                |
%         .- C -.  \)     _                 |                _
%        // = = \\,/      |                 |                |
%       () =    = ()      |                 |                |
%        \   ="=           |                 | 1              |
%        /)(    )          |                 |                |
%          // \\           |    1 - 0.79     |                |
%         //   \\          |                 |                |
%        ()     ()         |                 |                |
%        ||     ||         |                 |                |
%        |'     '|         |                 |                |
%      ==" B "==           _                 _                _
```

Figure 5.4: Patch extraction for the KECK gesture dataset - shows a snippet of the documentation in the Matlab-code performing the patch extraction. The patches extracted are shown in Figure 5.3.

5.2 Per Video Evaluations

The system was evaluated with both methods of sequence comparison (AL-DTW and SPRING) on introduced datasets, which will be discussed in the following.

In general it has to be mentioned that there are some differences in evaluating for a given video in contrast to a live stream. One difference is, that while evaluating a video the system can acquire a new frame at any time. While when using a live camera the system has to be fast enough to capture and process frames at a sufficiently small intervals. Another difference is that a video has an end and a beginning while a stream is pictured as an endless source of data.

5.2.1 Action Classification by AL-DTW

Firstly the system was evaluated using the AL-DTW algorithm (Section 4.8.1) for repetition detection. Each video was analyzed like a life stream with a frame rate around 5 frames per second (fps). Such was done for 5 different scales. Therefore all the temporal analysis of costs like proposed in 4.8.1

was deactivated and simply the action according to the minimal AL-DTW-cost was used. This results where captured and sorted according to their costs. The classification result which was most frequent within the k first of the sorted results was treated as per video classification result (this is a k-nearest neighbor evaluation). The results as shown in column 2 of Table 5.1 proved the system to be generalizing. Such evaluation does not evaluate the repetition detections for their temporal quality. All experiments which where made to evaluate this temporal quality failed, meaning around half of the repetitions where wrongfully detected. Only watching the cost minimum and analyzing for temporal local minima as described in Section 4.8.1 turned out to be not robust enough for the purpose of repetition detection. However, even if this does not prove the system to be a repetition detector, such is still a per video classification which can be compared to the results of others. Due to its low frame rate the algorithm was failing at action classification in live cameras streams. The low frame rate caused a temporal under sampling of the camera stream which is another reason why an alternative (the SPRING algorithm) had to be found.

The number of prototypes is 1382 the number of base vectors is 100 per feature (HOG HOF LBP LBFP). The k for the k-Nearest-Neighbors Algorithm is 4. The minimal cost for the later half of each video is analyzed for 7 scales. And the according action states the result. An overall recognition rate of 96.67% The according confusion matrix is shown in Figure 5.5.

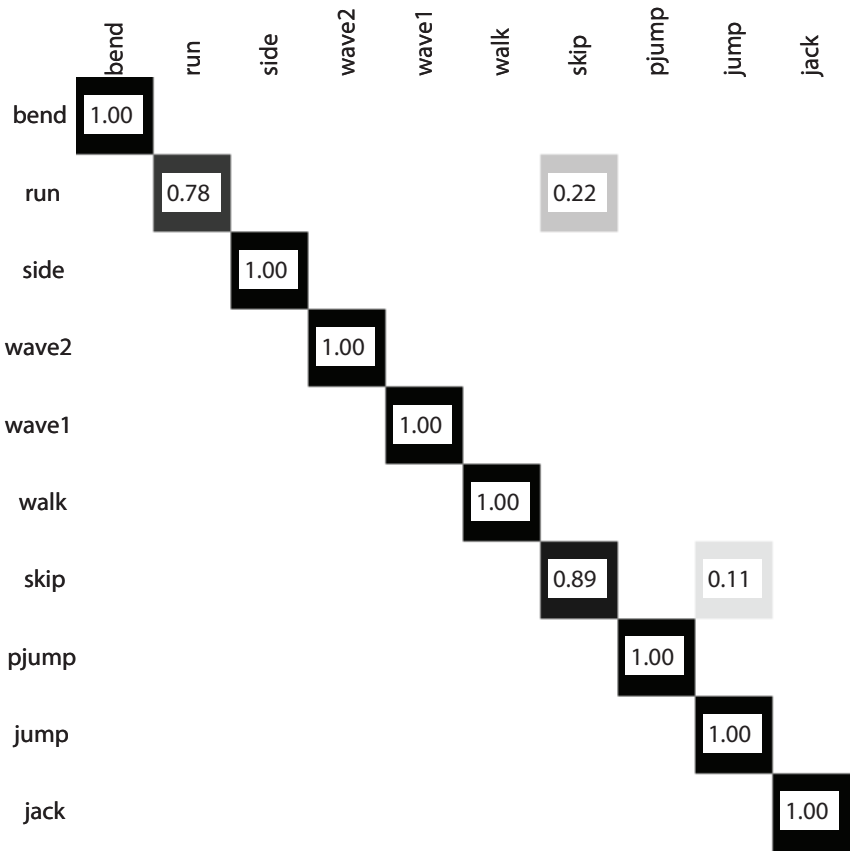


Figure 5.5: Confusion matrix for the Weizmann dataset in case of action recognition without repetition recognition by AL-DTW - The results are satisfying and the overall recognition rate is 96.67%

For the KTH-S1 dataset the evaluation was performed with about 945 leaves prototype-tree (dependent on the person left out) and 100 NMF-base vectors for each of the four features (HOG, HOF, LBP, LBFP) resulting in patch descriptors of 400 elements. The recognition rate (the number of videos correctly classified) is 87.33%. The according confusion matrix is shown in Figure 5.6. The KECK Gesture dataset was not analyzed by AL-DTW.

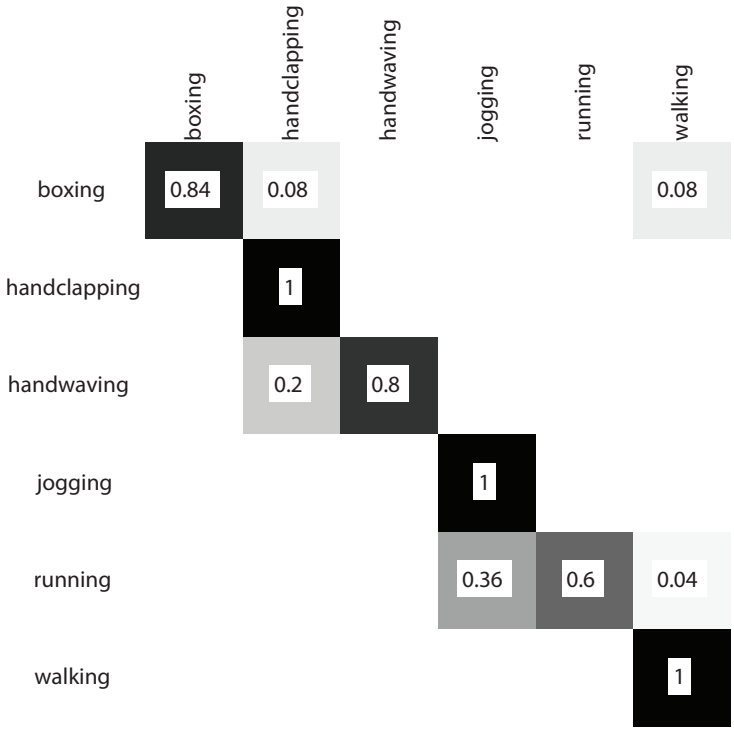


Figure 5.6: Confusion Matrix for the KTH S1 Dataset in case of action recognition without repetition recognition by AL-DTW - The overall detection rate is 87.33%. The confusion of running and jogging is understandable, since these two actions are very similar in nature.

5.2.2 Repetition Detection and Action Classification by SPRING

The videos of each of the datasets are not only containing the action performed but also noise. A person is, for example, standing and waiting before performing the next action. In classical video based action-recognition like performed in the previous section these parts of the video are ignored. Sometimes actors also walk into the scene at the beginning. To reduce the effect of such behavior but still evaluating on streaming data, while gaining comparable results again a k-nearest-neighbor was employed. In contrast to the AL-DTW-costs where a cost was generated at each frame, the SPRING algorithm outputs only reports when a repetition was detected. And the k-nearest-neighbor selects the most frequent report. The value of k is chosen to be eight for the Weizmann dataset and five for the KTH and KECK dataset. The results of all these evaluations are shown in Table 5.1. The according frame-rates (fps) where accomplished on a personal computer containing a Core2 Duo 2.66GHz CPU with a GForce 9800 GT GPU running a 32 bit system. This is no state of the art machine. An additional gain in speed can be assumed if a faster system is used.

The according confusion matrices are shown in Figure 5.7 for the Weizmann dataset, in Figure 5.9 for

the KTH Gesture dataset and Figure 5.8 for the KECK dataset.

The k of the k-nearest neighbour was eight for the Weizmann dataset and five for the KTH and KECK dataset. The number of prototypes are listed in Table 5.1.

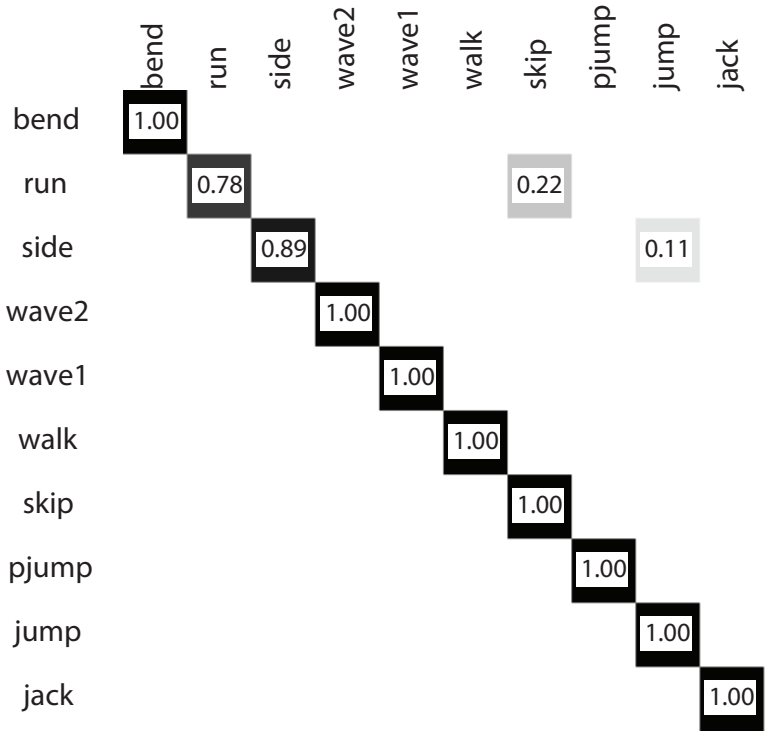


Figure 5.7: The confusion matrix for the Weizmann dataset by repetition detection by the SPRING algorithm - The results are reasonable and the overall recognition rate was 96.7%

	Proposed				Recognition Rate [%]		
	SPRING			AL-DTW	Roth09	Lin09	Yao09
	Num. Protot.	fps	Repetition Rec.	Action Rec.			
Weizmann	1461	14.23	96.7	96.7	94.2	100	-
KECK	883	10.12	73.3	-	-	95.24	-
KTH S1	915	11.51	78.6	87.33	88.1	98.83	90.1

Table 5.1: Results for evaluation on the Weizmann, the KTH database and a subset of the KECK database - The works compared to are Roth09 [Roth et al., 2009], Lin09 [Lin et al., 2009] and Yao09 [Yao and Zhu, 2009].

Our results are not outperforming any of the related work on standard datasets but they can be computed near real time. Compared to the Lin *et al.* [Lin et al., 2009] our approach does not perform any tracking. Tracking does not conflict with the real-time requirement, because once an actor is tracked only parts of the image could be evaluated, but it is deliberately left out in this work. This explains the modest results for the KECK dataset since in the KECK dataset the backgrounds and the scenes in the training set are not comparable to the test-set. Lin employ a generic human detector for actor-detection they cite Dalals HOG detector [Dalal and Triggs, 2005]. It was also tried to use such a generic human detector to

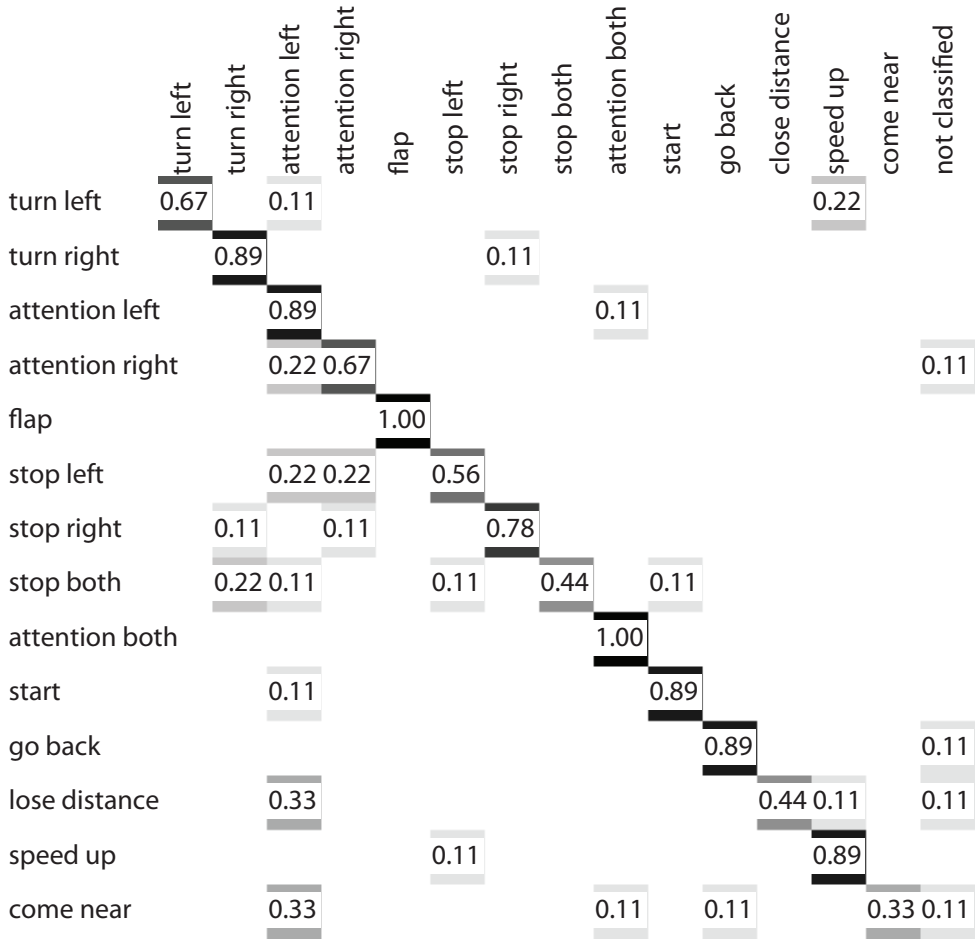


Figure 5.8: The confusion matrix for the KECK dataset by repetition detection by the SPRING algorithm - The many confusions of come near and stop both with other actions, are explainable by looking at their patches in Figure 5.3. They are not where significant sequences.

overcome this problem, but with focus on real time, leading to the GPU based generic human detector of the Robot Operating System (ROS) [Quigley et al., 2009]. Still the results where the same failed detections like shown in Figure 5.10. Another problem is that some actions like “boxing” in the KTH dataset (see Figure 5.2) are not suitable as a gesture at all, because they simply can be performed in many very different ways and the characteristic is not based on such strict sequence of prototypes. Such could be interpreted as a an action like boxing is an ongoing activity rather than an action with a clear beginning and end.

5.3 Temporal Analysis

To validate the system for its temporal accuracy, and the ability to recognize different actions performed after one another, test data was generated from the original Weizmann videos. Therefore randomly chosen actions where selected, and one to three repetitions of it where appended to the test sequence. Resulting in a video containing multiple actions, where each is repeated one to three times. This video was always constructed by using the person, which has been left out during training. For each person 20

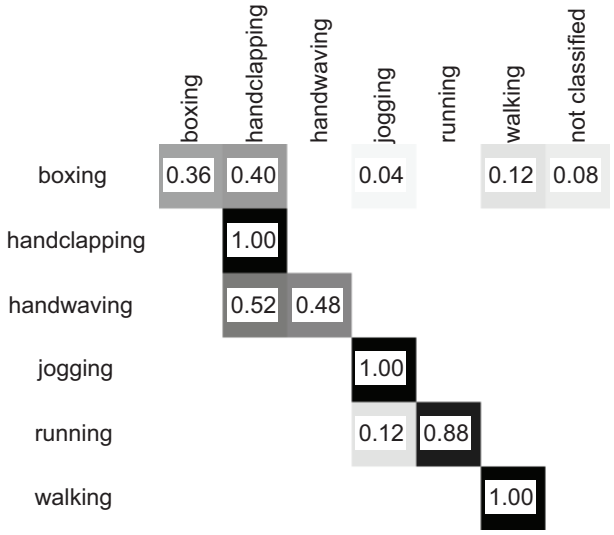
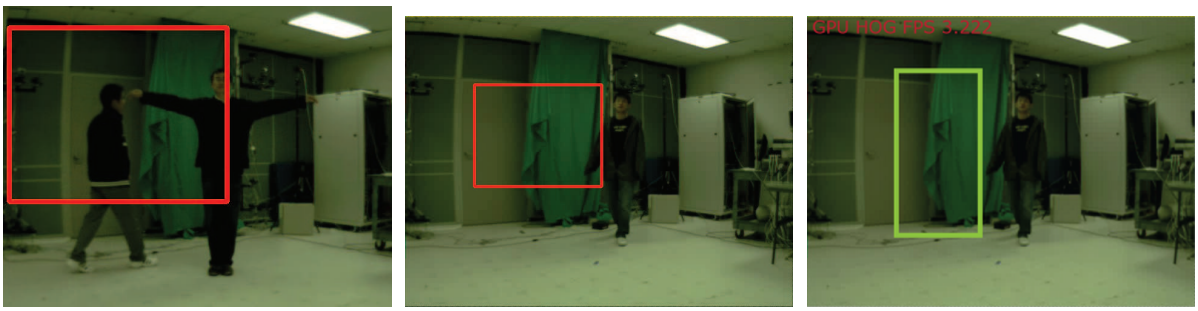


Figure 5.9: Confusion matrix for KTH-S1 classified by SPRING - The boxing action is confused a lot and without any pattern. A possible interpretation is that boxing is more of an activity than a gesture. Therefore the signature sequence of poses is not descriptive enough. The confusion of handwaving and handclapping happens for unknown reasons.

of these videos where created per person. The classification results and their temporal alignment for the first ten are shown in Figure 5.11.

The analysis was performed of the results was performed on a per frame basis and a semantical basis. To express the results of Figure 5.11 in numbers the Levenstein-distance [Levenstein, 1966] is employed. Given two sequences of symbols (in our case actions), one is the annotated sequence and one the classification result action-sequence the Levenstein-distance computes the number of edit operations (which are "add symbol", "remove symbol" and "exchange symbol") necessary to transform the classification result into the annotated action-sequence. The average normalized (by the number of symbols) Levenstein-distance for all of these randomized experiments is 0.345. As we can see in the graphic this means that failures are common. The number of correctly classified frames to the number of frames equals 0.74.

This results are hard to interpret, but it is obvious that there is space for improvement in future works. It



(a) Person passing

(b) False detection of the curtain as person.

(c) False detection by the GPU HOG.

Figure 5.10: Problematic actor-detections in the KECK dataset - (a) shows a person passing which is detected as actor. (b) is a detection of a curtain as person by the proposed system, while (c) shows the same scene with a detection by the ROS version of OpenCVs HOG-based pedestrian detector.

also shows that for such task of repetition detection the Weizmann dataset is already a hard task. Results of repetition-detection for the other datasets will obviously be inferior, since their classification results are already inferior.

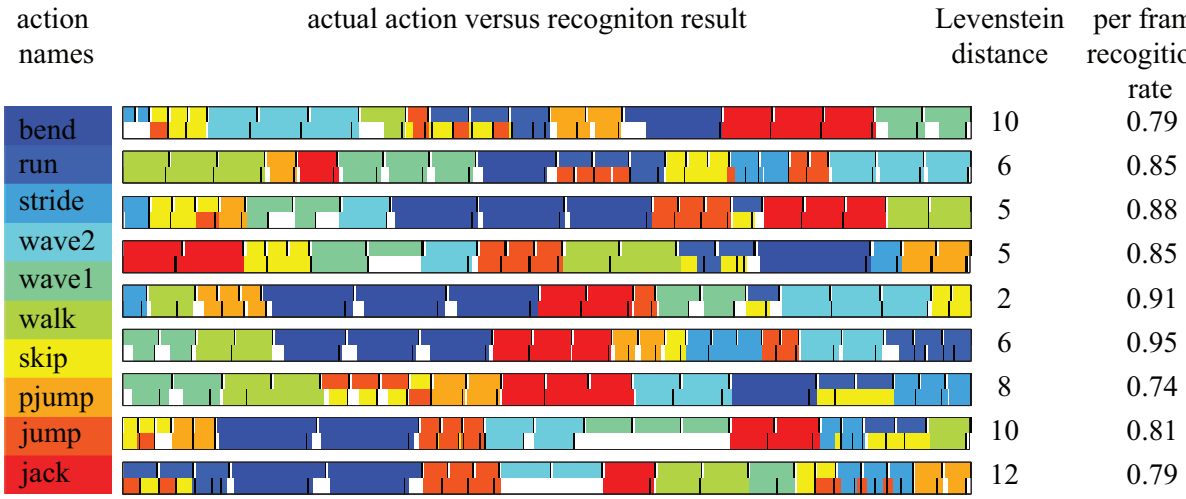


Figure 5.11: Results for temporal analysis of randomly generated videos composed of Weizmann actions - Each containing randomly chosen actions performed for varying number of times in a row. Each row consists of one color-bar indicating the annotated action classification by the colors corresponding to Figure 5.1 and another presenting the accomplished classification results. The black lines indicate the action ends while white represents no classification at all.

Nevertheless it turns out that if the number of actions is significantly reduced this approach can still be used for real-world applications as shown in Section 6.

Chapter 6

Actions Controlling Applications

To control applications the concept, that the performance of one repetition of an action is equivalent to pressing a key, is realized by employing a virtual keyboard. Dependent on which action was performed different keys are pressed. Therefore the runtime-system is extended by virtual keyboard interface realized by AutoHotkeyDll which is a forge of the AutoHotkey[1]. The Original AutoHotkey is a Windows tool for automation of repetitive tasks. The core component is a interpreter for a AutoHotkey specific scripting language allowing to start programs, change window focus, and to generate mouse and keyboards events. The AutoHotkeyDll is a forge of AutoHotkey which can be integrated into self written applications like the runtime-system proposed in this work. The runtime-system is triggering the execution of specific scripts for each action-repetition detected. By exchanging the scripts a wide palette of applications can be controlled. Which script is invoked by which action is controlled by a configuration file (see Figure 6.1). Each of the scripts is switching focus to the target application and pressing a specific key, or a sequence of keys. In the following some example configurations are presented.

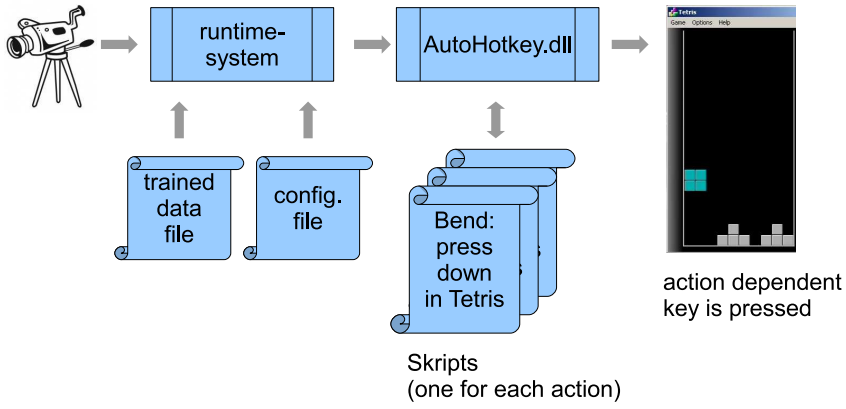


Figure 6.1: Component interaction for invoking a virtual keystroke - The runtime-system is analyzing the live video stream, captured by the camera, therefore it uses the information acquired during training (the trained data file). Dependent on the configuration it invokes the execution of the AutoHotkey-scripts. The configuration file contains the mapping from action to keystroke. Each script contains the information of how to control the target application.

[1] http://www.autohotkey.com/

In Figure 6.2 an experiment is shown, where a video composed of different Weizmann actions (one repetition of each) controls the Windows WordPad. The composed video shows the leave out person. This is a demo under perfect conditions, since the system was trained with similar backgrounds. Therefore it works fairly well. One thing that can be observed is, that walking is detected twice (as shown in

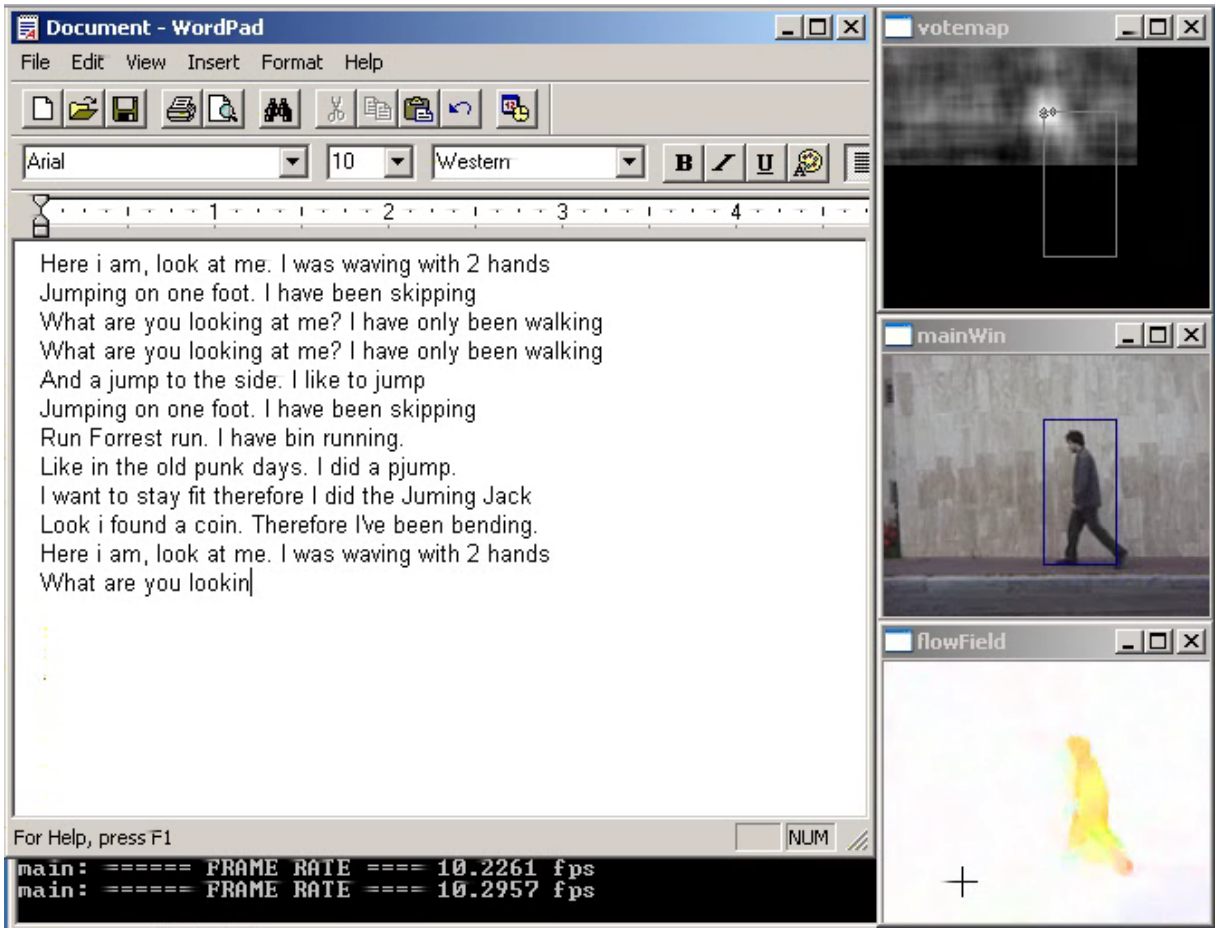


Figure 6.2: Screenshot of controlling the Windows WordPad by Weizmann actions - Each action of the Weizmann dataset writes a different sentence in Word Pad. The video running through contains only one repetition per action. Therefore the 4th row is caused by a false positive for the walking action.

Figure 5.1). The video only showed it once. The reason is, that walking is defined as making one step with the foot facing the camera (the front foot) and one with the other foot (the back foot). In retrospect this might have been a failure. Even if this reflects the idea that a human would not call it “walking”, if a person always uses the same foot first, the difference in the image sequence between moving the front foot or the back foot is that small, that the system can not differentiate. A better repetition definition would be to define “making one step” (as gesture) and not walking (as activity).

The data trained for the Weizmann dataset only, turned out to be generalizing enough to be able to control a video game (with four actions) with a live camera without any bootstrapping or background subtraction. The action to key mapping in this example is shown in Table 6.1.

Action		Key	Meaning
bend	↔	down	Set stone down
stride	↔	left	Move stone left
wave2	↔	right	Move stone right
jump	↔	space	Turn stone by 90°

Table 6.1: Action to keyboard mapping for Weizmann actions controlling Tetris - For the definition of a repetition of one of these actions see Figure 5.1.

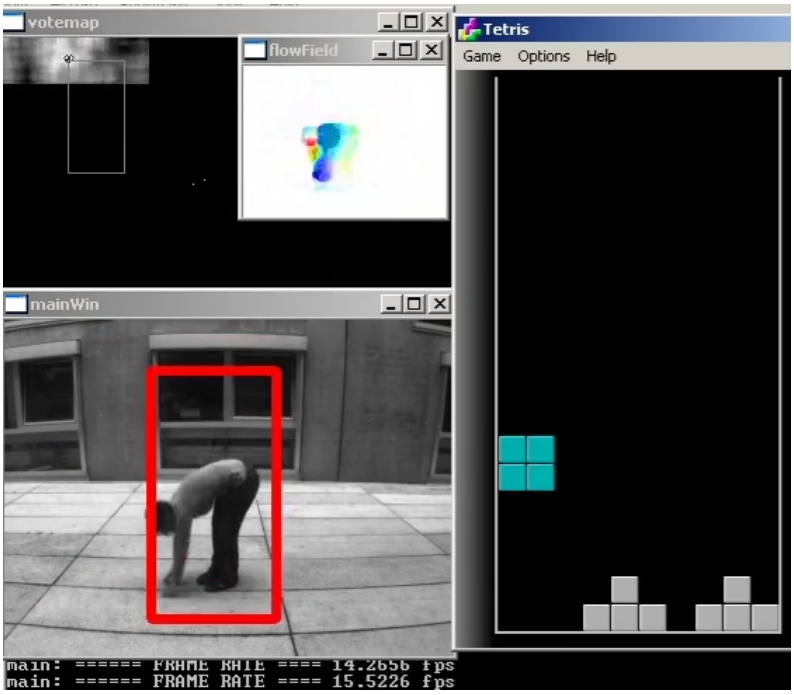


Figure 6.3: Screenshot for controlling the Tetris Game by Weizmann actions - The system controlling Tetris (the bend action will put down the stone.)

This setup worked fairly well. It is important, that the background does not contain elements which are detected as humans. For example jackets hung up on a coat rack or sometimes even chairs can be detected as human body. If such is not case it works fairly well. We are aware of the fact, that the actions are not very intuitive in their meaning. A simple wave action with either the left or the right hand are more user friendly. Such could be realized by retraining the system to be distinguishing them as different actions. But as a proof of concept it was more important to be able to distinguish as many actions as possible. Another effect which was observed is that users have more fun, the more complicated the actions are.

When choosing the mapping it is important that the action which is setting down the stone is unlikely to be performed by accident. “walking” or “stride” are actions which are caused by accident (when the user feels the urge to move). Despite this fact, “stride” was used in this example-configuration but is reversible by performing a “wave2”.

One of the most obvious impressions when using the system, is that it seems to have a lot of latency. This is actually not the case. But the user needs to get used to the fact that an action is not causing an effect until the gesture is finished. Such is a very different experience compared to a keyboard or joystick as human-interface. This impression is emphasized by the fact, that some repetition definitions are not fitting the "perceptive hotspot". For example a "bend" action-repetition, as defined in our system, is starting with an upright standing person and ending with an upright standing person. The "perceptive hotspot" would be the point where the person touches the ground and not the upright standing position. But the system detects the bend-repetition not until the person is standing again. This can be seen in Figure 6.4.

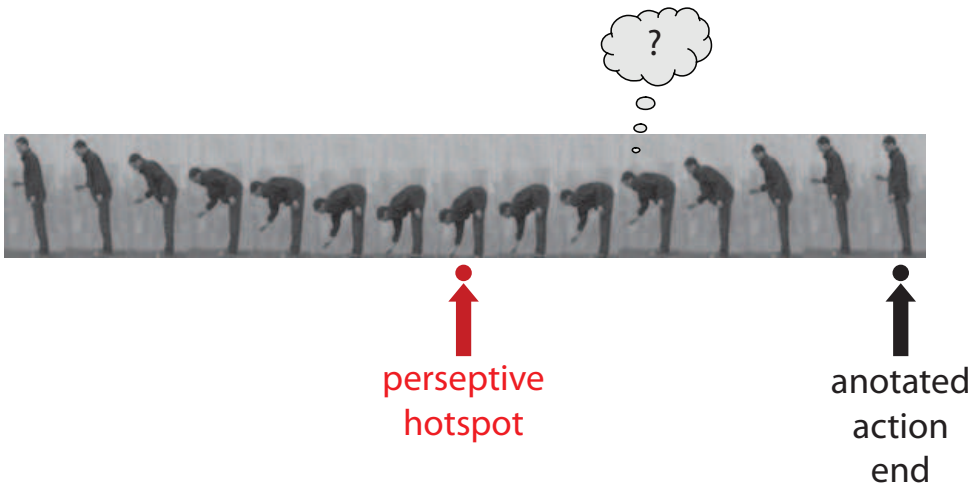


Figure 6.4: Bend action "percetive hotspot" versus actual action end - After the "perceptive hotspot" is passed the user starts to ask himself: "What is going on? Why was there no effect?"(Cite of an independent test user). Until the annotated action end is reached and the stone is put down.

The data trained for the KECK Gesture dataset was used to control Tetris as well. The mapping in Table 6.2 was defined. A screen-shot is shown in Figure 6.5.

Action		Key	Meaning
speed up	↔	down	Set stone down
turn left	↔	left	Move stone left
attention right	↔	right	Move stone right
flap	↔	up	Turn stone by 90°

Table 6.2: Action to keyboard mapping for KECK gestures controlling Tetris - For the definition of a repetition of one of these actions see Figure 5.3.

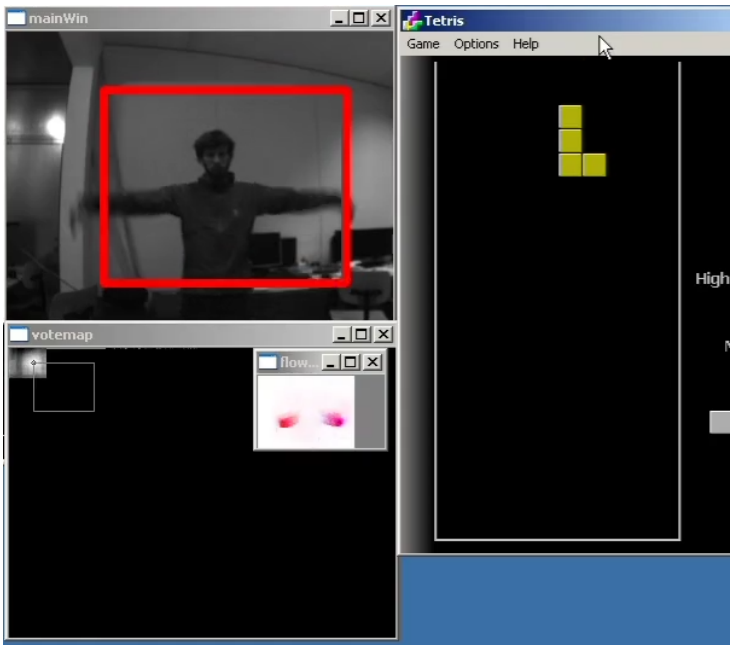


Figure 6.5: Screenshot for controlling Tetris with KECK gestures - “flap” is performed at this moment. It causes to rotate the stone.

In case of the KECK actions, the repetition-definition is more appropriate than the one in the previously discussed Weizmann controlled Tetris. The effect of gestures is more intuitive, but due to the problems which have been pointed out in Section 5.2 with the KECK dataset, the actions are not detected as well. It also seems, that the system is not as independent from the background. The actor detector based on the patch definition is not performing as well as the actor detector for the Weizmann dataset (shown in Figure 5.10). Users generally prefer the KECK Gestures over the Weizmann actions for the purpose of controlling Tetris.

Chapter 7

Conclusion

In this work a pipeline of modules has been presented which is able to perform video based human body motion action-classification near real time. Therefore the two types of features, namely the Histogram of Gradients and the Local Binary Patterns, have been implemented on graphics card to exploit its parallel computing capability. We showed how to adapt these features to analyze motion information, leading to the Histograms of Flow Orientations and the Local Binary Flow Patterns. As motion representation a dense optical flow field was used. Non Negative Matrix Factorization was applied to the combination of the previously calculated features to reduce them in dimension while keeping their expressiveness.

A strategy for minimizing CPU GPU data transfer was presented, it works as follows: A linear Support Vector Machine was used to detect the actor in a camera image. It was shown how to combine the evaluation of a linear SVM and an NMF to a single operation. Up to this point all calculations have been performed on the GPU. Only the features corresponding to the actor-detection result are transfered to CPU memory.

This features where used to compute a per-frame-prototype still representing pose and motion. Such was performed by traversing a hierarchy of k-means clusters.

The stream of prototypes was now compared to the known action-specific candidate sequences of prototypes in an online manner. Therefore a stream specialized version of Dynamic Time Warping, called SPRING, was used to find a subsequence-match inside the prototype-stream. Dependent on the label of the candidate sequence the action was classified. Additionally the alignment of the subsequence was determined. Which allowed for fast and fairly accurate sub-sequence matching.

The system was evaluated on the Weizmann the KTH and KECK Gesture human body action datasets leading to competitive (but worse) results as previous publications gained, while keeping up with the real-time requirement. Additionally the system was evaluated on videos composed of different actions to analyze its ability to be used as keyboard substitute.

To train and test the system additional temporal annotation has been added to the KTH and Weizmann dataset. This annotation is the beginning-frame and end-frame of a single repetition of an action. This information corresponds to a previously defined repetition definition, of what exactly constitutes a single repetition of an action.

Finally, proof of concept was given, by using the pipeline as keyboard substitute, in order to control

a game (and other applications) with actions. Two problems where identified. One is that the defined action-ends often differ from an intuitive action-end. The other, that some actions are performed involuntary. We postulate that a good action to be used for gaming-applications (with repetition detection) should be more of a gesture than an activity, with an intuitive end.

Prospect

To actually proof the systems ability to be used as HCI a complete HCI-user-study has to be performed. Only this way the quality of different actions to be used as a keyboard substitute can be evaluated.

The incorporation of tracking into the system is a logical consequence from the lessons learned during evaluation. A particle system taking advantage of the calculated votemap comes in mind. The cost of the DTW path of the subsequence-match can be used to analyze the tracks. If over a longer time period no subsequence-matches can be found for a track, and it is not moving, it can be considered to be background. This information can be used to retrain the detector during evaluation. Such would reduce the negative effect of human like structures in the scene.

To improve speed, the traversal of the k-means-hierarchy can also be performed on the GPU. This would allow for fast analysis of multiple tracks without losing the real-time capability, and the classification of actions performed by multiple persons respectively.

Bibliography

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer, 1st ed. 2006. corr. 2nd printing edition. (Cited on pages 2, 3, 25 and 28.)

[Blank et al., 2005] Blank, M., Gorelick, L., Shechtman, E., Irani, M., and Basri, R. (2005). Actions as space-time shapes. In *The Tenth IEEE International Conference on Computer Vision (ICCV'05)*, pages 1395–1402. (Cited on page 3.)

[Bradski, 2000] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. (Cited on page 13.)

[Cedras and Shah, 1995] Cedras, C. and Shah, M. (1995). Motion-Based recognition: A survey. *IMAGE AND VISION COMPUTING*, 13:129—155. (Cited on page 14.)

[Chan, 2007] Chan, S. S. (2007). Fast DTW: toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580. (Cited on pages 13, 33 and 36.)

[Chang and Lin, 2011] Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. (Cited on page 29.)

[Choi, 2008] Choi, S. (2008). Algorithms for orthogonal nonnegative matrix factorization. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1828–1832. (Cited on page 27.)

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297. (Cited on page 29.)

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, page 886. (Cited on pages 13, 21, 22 and 74.)

[Dollár et al., 2005] Dollár, P., Rabaud, V., Cottrell, G., and Belongie, S. (2005). Behavior recognition via sparse spatio-temporal features. In *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*, page 65–72. (Cited on page 14.)

[Efros et al., 2003] Efros, A. A., Berg, A. C., Mori, G., and Malik, J. (2003). Recognizing action at a distance. In *Computer Vision, IEEE International Conference on*, volume 2, page 726, Los Alamitos, CA, USA. IEEE Computer Society. (Cited on page 12.)

[Gorelick et al., 2005] Gorelick, L., Blank, M., Shechtman, E., Irani, M., and Basri, R. (2005). Actions as space-time shapes. *IN ICCV*, 2005:1395—1402. (Cited on pages 11 and 67.)

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441. (Cited on page 25.)

[Kale et al., 2004] Kale, A., Sundaresan, A., Rajagopalan, A. N., Cuntoor, N. P., Roy-Chowdhury, A. K., Kruger, V., and Chellappa, R. (2004). Identification of humans using gait. *Image Processing, IEEE Transactions on*, 13(9):1163–1173. (Cited on page 15.)

[Ke et al., 2007] Ke, Y., Sukthankar, R., and Hebert, M. (2007). Event detection in crowded videos. In *IEEE International Conference on Computer Vision*, volume 23, page 38–41. (Cited on page 11.)

[Laptev, 2005] Laptev, I. (2005). On space-time interest points. *International Journal of Computer Vision*, 64(2):107–123. (Cited on pages 14 and 15.)

[Laptev et al., 2008] Laptev, I., Marszalek, M., Schmid, C., and Rozenfeld, B. (2008). Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. (Cited on page 24.)

[Laurel, 1990] Laurel, B. (1990). *The Art of Human-Computer Interface Design*. Addison-Wesley Professional, 1 edition. (Cited on page 4.)

[Lee and Seung, 1999] Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791. (Cited on pages 25 and 26.)

[Levenstein, 1966] Levenstein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*. (Cited on page 76.)

[Lin et al., 2009] Lin, Z., Jiang, Z., and Davis, L. S. (2009). Recognizing actions by shape-motion prototype trees. In *Proceedings of the International Conference On Computer Vision (ICCV'09), Kyoto, Japan*, page 1–8. (Cited on pages 11, 13, 31, 32, 33, 58, 63, 67 and 74.)

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110. (Cited on pages 13 and 21.)

[Müller, 2007] Müller, M. (2007). *Information Retrieval for Music and Motion*. Springer, 1 edition. (Cited on pages 13, 36 and 37.)

[Mauthner et al., 2011] Mauthner, T., Roth, P., and Bischof, H. (2011). Temporal feature weighting for prototype-based action recognition. *Computer Vision–ACCV 2010*, page 566–579. (Cited on page 13.)

[Muybridge, 1955] Muybridge, E. (1955). *The Human Figure In Motion*. Dover Publications, Mineola, NY. (Cited on page 1.)

[Nater et al., 2010] Nater, F., Grabner, H., and Van Gool, L. (2010). Exploiting simple hierarchies for unsupervised human behavior analysis. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, page 2014–2021. (Cited on pages 4, 13 and 31.)

[Nistér and Stewénius, 2006] Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. *IN CVPR*, 2:2161—2168. (Cited on page 21.)

[NVIDIA, 2009a] NVIDIA, C. (2009a). C programming best practices guide. *Cuda Toolkit*, 2. (Cited on pages 45 and 46.)

[NVIDIA, 2009b] NVIDIA, C. (2009b). CUDA programming guide. *NVIDIA: Santa Clara, CA*, 2.3.1. (Cited on pages 13 and 45.)

[Ojala et al., 2000] Ojala, T., Pietikäinen, M., and Mäenpää, T. (2000). Gray scale and rotation invariant texture classification with local binary patterns. In *Computer Vision - ECCV 2000*, pages 404–420. (Cited on page 19.)

[Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to points in space. *Philosophical magazine*, 2:559–572. (Cited on page 25.)

[Pentland et al., 1991] Pentland, M. T., A., Turk, M., and Pentland, A. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86. (Cited on pages 15 and 25.)

[Polana and Nelson, 1997] Polana, R. and Nelson, R. C. (1997). Detection and recognition of periodic, nonrigid motion. *International Journal of Computer Vision*, 23(3):261–282. (Cited on page 14.)

[Prisacariu and Reid, 2009] Prisacariu, V. A. and Reid, I. (2009). fastHOG- a real-time GPU implementation of HOG technical report no. 2310/09. (Cited on page 13.)

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*. (Cited on pages 13 and 75.)

[Reddy et al., 2009] Reddy, K. K., Liu, J., and Shah, M. (2009). Incremental action recognition using feature-tree. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1010–1017. IEEE. (Cited on pages 11 and 12.)

[Rodriguez et al., 2008] Rodriguez, M. D., Ahmed, J., and Shah, M. (2008). Action MACH a spatio-temporal maximum average correlation height filter for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, pages 1–8. IEEE. (Cited on page 11.)

[Roth et al., 2009] Roth, P. M., Mauthner, T., Khan, I., and Bischof, H. (2009). Efficient human action recognition by cascaded linear classification. *1st IEEE Workshop on Video-Oriented Object and Event Classification*. (Cited on pages 12, 21, 26 and 74.)

[Sakurai et al., 2007] Sakurai, Y., Faloutsos, C., and Yamamuro, M. (2007). Stream monitoring under the time warping distance. In *2007 IEEE 23rd International Conference on Data Engineering*, page 1046–1055. (Cited on pages 7, 13, 36, 58, 62 and 63.)

[Schuldt et al., 2004] Schuldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: A local SVM approach. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3-Volume 03*, page 36. (Cited on pages 11 and 67.)

[Sonka et al., 1999] Sonka, M., Hlavac, V., and Boyle, R. (1999). Image processing, analysis, and machine vision second edition. *International Thomson*. (Cited on page 17.)

[Thurau and Hlavac, 2008] Thurau, C. and Hlavac, V. (2008). Pose primitive based human action recognition in videos or still images. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Anchorage, AK, USA. (Cited on page 11.)

[Turaga et al., 2008] Turaga, P., Chellappa, R., Subrahmanian, V. S., and Udrea, O. (2008). Machine recognition of human activities: A survey. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1473–1488. (Cited on pages 1, 4, 8 and 14.)

[Wang et al., 2009] Wang, X., Han, T. X., and Yan, S. (2009). An HOG-LBP human detector with partial occlusion handling. In *2009 IEEE 12th International Conference on Computer Vision*, pages 32–39. IEEE. (Cited on pages 13 and 21.)

[Weinland and Boyer, 2008] Weinland, D. and Boyer, E. (2008). Action recognition using exemplar-based embedding. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 0, pages 1–7, Los Alamitos, CA, USA. IEEE Computer Society. (Cited on page 11.)

[Yao and Zhu, 2009] Yao, B. and Zhu, S. C. (2009). Learning deformable action templates from cluttered videos. In *Computer Vision, 2009 IEEE 12th International Conference on*, page 1507–1514. (Cited on pages 11, 12 and 74.)

[Yilmaz and Shah, 2005] Yilmaz, A. and Shah, M. (2005). Actions sketch: A novel action representation. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, page 984–989. (Cited on page 15.)

[Yu et al., 2010] Yu, T., Kim, T., and Cipolla, R. (2010). Real-time action recognition by spatiotemporal semantic and structural forests. In *Procedings of the British Machine Vision Conference 2010*, pages 52.1–52.12, Aberystwyth. (Cited on pages 11 and 12.)

[Zach et al., 2007] Zach, C., Pock, T., and Bischof, H. (2007). A duality based approach for realtime TV-L1 optical flow. In *Pattern Recognition: 29th DAGM Symposium, Heidelberg, Germany, September 12-14, 2007, Proceedings*, page 214. (Cited on page 18.)

[Zelnik-Manor and Irani, 2001] Zelnik-Manor, L. and Irani, M. (2001). Event-Based analysis of video. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 2, page 123, Los Alamitos, CA, USA. IEEE Computer Society. (Cited on page 15.)

[Zhang and Nevatia, 2008] Zhang, L. and Nevatia, R. (2008). Efficient scan-window based object detection using GPGPU. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. CVPR Workshops 2008*, page 1–7. (Cited on page 23.)

[Zhong et al., 2004] Zhong, H., Shi, J., and Visontai, M. (2004). Detecting unusual activity in video. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 2, pages 819–826, Los Alamitos, CA, USA. IEEE Computer Society. (Cited on pages 13, 14 and 15.)

[Zhou et al., 2008] Zhou, F., Torre, F., and Hodgins, J. K. (2008). Aligned cluster analysis for temporal segmentation of human motion. In *8th IEEE International Conference on Automatic Face & Gesture Recognition, 2008. FG '08*, pages 1–7. IEEE. (Cited on page 12.)

[Zhou and Torre, 2009] Zhou, F. and Torre, F. D. l. (2009). Canonical time warping for alignment of human behavior. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, page 2286–2294. (Cited on page 12.)