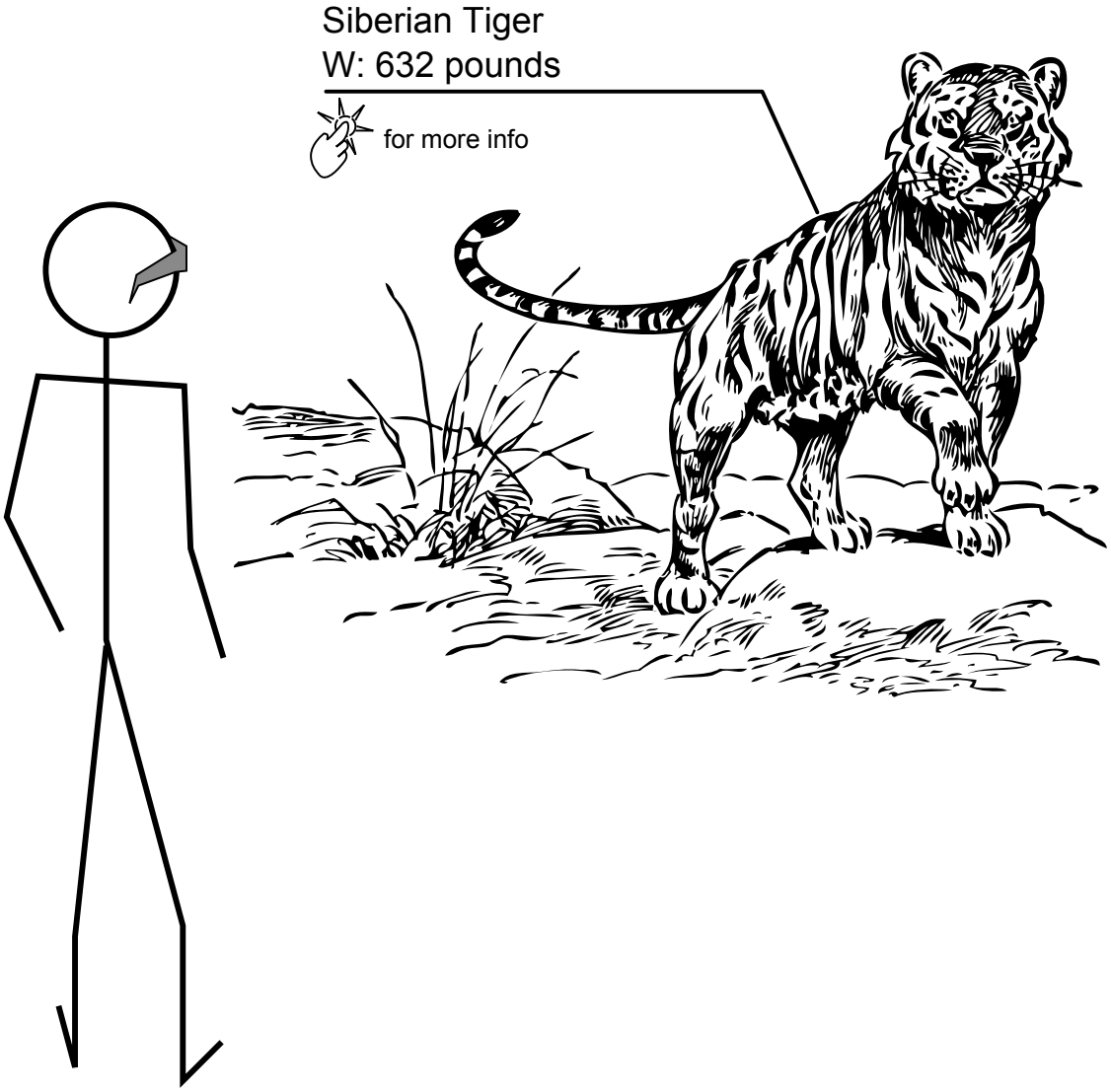
Analysis and Synthesis of Images

Animal detection and Action Recognition Using Histograms of Oriented Gradients and dense Optical Flow



Vision, Graphics and Interactive Systems
10th semester Masters Thesis 2011, Group 1023

AALBORG UNIVERSITET

Disney Research, Pittsburgh

Det Ingeniør-, Natur- og Sundhedsvidenskabelige Fakultet - Elektronik og Elektroteknik
Fredrik Bajers Vej 7
9220 Aalborg Ø
http://esn.aau.dk

Titel: Animal detection and Action Recognition Using Histograms of Oriented Gradients and dense Optical Flow

Theme: Analysis and Synthesis of Images
Specialisation: Vision, Graphics and Interactive Systems
Project period: 1/9, 2010 - 31/5, 2011

Group number: 1023

Students:

Thomas Balling Sørensen
Henrik Møss Christoffersen

Supervisor:

Leonid Sigal
Thomas B. Moeslund

Synopsis:

The increase in advanced handheld devices, such as smartphones and tablets have opened up to many new ways of letting visitors interact with different kinds of exhibits. Attractions such as museums and sightseeing tours often take advantage of the fact that most visitors have mobile phones, which they can use to get selective information of what they are looking at or play minigames related to the exhibit.
In this project we explore one way to improve the visitor experience in a zoological park, by augmenting information directly into the view of the user, based on what is being observed. To do this we use a head-mounted-display equipped with cameras, connected to a PC. Our goal is to detect and recognize specific animals and extract motion data to identify what action each animal is performing. The information obtained is then processed and presented to the user in a meaningful way.
We implement a cascaded SVM classifier based on HOG descriptors. The first stage is used to identify the animal from its appearance using images from the cameras. In the second stage, motion data is obtained from a GPU accelerated optical flow algorithm and the resulting motion frames (maps) are used to identify the action. We use motion flow over several frames to encode motion relations that identify different actions, to improve performance. We also implement a simple method to limit the influence of camera motion on the motion frames.
In order to verify the system, each stages of the cascaded SVM classifier is tested individually. Our system prove to be good at detecting elephants and recognize its two actions. Due to the small amount of animal actions in the optained elephant dataset, it was chosen to test the system against the KTH dataset containing six different actions. The action recognition results on this dataset were less convincing and suggests the need of improvement.

Copies: 4
Number of pages: 106
Attachments: 1 CD
Finished: 31/5 2011

Preface

This report has been composed during our stay in Pittsburgh, U.S.A. in collaboration with Carnegie Mellon University and Disney Research Pittsburgh. Our stay in Pittsburgh began in November 2010 and lasted until June 2011. From November until the end of January our work was primarily focused on helping on improving the calculation speed of a project, which our supervisor Leonid Sigal was working on at the time. The work on this project is described in appendix A. This happened while we were waiting for the necessary hardware to start our main project of recognizing animals and their actions based on motion and appearance. The hardware we were waiting for is a commercial stereo vision head-mounted-display from Vuzix, which is described in chapter 2.

We would like to thank Jessica Hodgins for allowing us the great opportunity to work with both DRP and CMU and Leonid Sigal for supervising our work and making this project possible. We would also like to thank Mykhaylo Andriluka for his advice on technical details and improvements and everyone at Disney Research Pittsburgh for welcoming us to their office and for making us feel at home.

The attached CD contain source code, working demos and examples of data used in this project.

Henrik Møss Christoffersen

Thomas Balling Sørensen

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Related Work 2

1.2.1 Augmented Reality 2

1.2.2 Object Recognition and Detection 3

1.2.3 Action Recognition 3

1.3 Contribution 4

I Analysis **5**

2 Platform **6**

2.1 Hardware 6

2.2 Interfaces 6

2.3 Features 7

2.4 Extensions 7

3 Problem definition **8**

3.1 Overall Requirements 8

3.2 System Requirements 9

3.2.1 Classification step 9

3.2.2 Augmentation step 10

3.3 System Limitations 10

3.4 Scope 12

3.4.1 Classification stage 12

3.4.2 Augmentation 12

4 System Definition **13**

4.1 Classification step 13

4.1.1 Choice of classifier 13

4.1.2 Choice of descriptor 14

4.1.3 Choice of features 15

4.1.4 Summary 16

4.2 Augmentation step 16

II Proposed Solution **19**

5 The classification step **20**

5.1 Cascade of classifiers 21

5.2 Motion maps 23

5.2.1 General optical flow 24

5.2.2 Motion estimation using Non-Local Total Variantion Requlariza tion . . . 27

5.2.3 Invariance to camera movements 30

5.2.4 Scaling of the motion vector maps 33

5.2.5 Multi-motion maps 35

5.2.6 Running average 37

5.3 Histogram of oriented gradients 37

5.3.1 Gamma/color normalization 38

5.3.2 Gradient computation 38

5.3.3 Orientation Binning 39

5.3.4 Normalization of the descriptor blocks 39

5.3.5 Collect and append HOG descriptors over the detection window 40

5.4 Support Vector Machine 41

5.4.1 Derivation of the SVM classifier 42

III Verification **50**

6 Methodology **51**

6.1 Verification data notes 51

7 Animal detection and recognition **52**

7.1 Choice of descriptors 53

7.2 Setup 53

7.3 Training and verification data 54

7.4 Results 56

8 Animal action recognition **60**

8.1 Choice of descriptors 60

8.2 Setup 61

8.3 Training and verification data 61

8.4 Results 64

9 Human recognition **67**

9.1 Choice of descriptors 67

9.2 Setup 67

9.3 Training and verification data 68

9.4 Results 70

10 Discussion **73**

IV Conclusion and Future work **77**

11 Conclusion **78**

12 Future Work **80**

V Appendix **83**

A Performance Optimization of HumanEva using OpenGL **84**

A.1 MATLAB code before optimization 85

A.2 Timing results before optimization 87

B Implementation of optimized functions **89**

B.1 Mesh rendering 89

B.2 Likelihood 91

B.2.1 Silhouette 93

B.2.2 Average 94
B.3 c_skinning and c_projection 97
B.3.1 c_skinning 97
B.3.2 c_projection 97
B.4 Misc OpenGL functions 98
B.4.1 gl_create_texture 98
B.4.2 gl_get_texture 99
B.4.3 gl_init and gl_quit 99

C Timing evaluation after optimization **100**

Bibliography **103**

Chapter 1

Introduction

1.1 Motivation

With the advance in technology in the past decade, it is becoming increasingly popular to find entertaining ways of incorporating information through interactive media devices in attractions of all sorts. The modern user is expecting to have information delivered in entertaining and practical ways, improving their experience when visiting attractions such as landmarks, museums and animal parks. When visiting the Empire State building in New York, it is possible to purchase access to a preprogrammed remote and a map with several landmarks marked on it. By entering a number of a landmark, visible from the top of the building, information about the landmark is delivered through a speaker build into the remote. The user can then observe the landmarks while being taught about their history and surrounding structures.

In many museums a user can interact with the exhibits, such, for example as switching prisms in a display of how light travels through telescopes, hearing about the movement of glaciers during the ice age, while watching a video of how the landscape is changed by this or seeing how much your stomp will affect a seismograph[1][2]. In animal parks similar exhibits are available, letting the visitor hear the many sounds of the animals they meet in the park, taking quizzes on what different expressions in a monkeys face mean or seeing how humans affect the natural habitats. Most of these exhibits are usually stationary and indoor, rarely in interaction with the live animal habitats in the park. At the live habitats, information about the animals are usually relayed to the user through informative signs placed along the enclosure. As a more mobile and interactive approach, London Zoo introduced a device called i-Track in 2008. The i-Track helped visitors navigate the zoo and let the visitor take quizzes, provided information on animals in the park and could play sound and video clips of animals from the zoo[3].

There is no doubt that increased interactivity with attractions through multimedia devices of all sorts, at static as well as live exhibits, is important to visitors and can be seen as an attraction in itself attracting more visitors. In 2007 Wellington zoo launched an advertisement campaign, placing special markers in their ads, which would show a 3D animal when watched through a

video phone using their free software package. The campaign allegedly lead to a growth of 32% in the visitors count of the zoo[4][5].

The most common information relay at live habitats in zoological parks consists of signs, describing the type of animals living there and the visitor often spend much time reading these, unable to observe the animals in the mean time. This is especially true for animal parks with roaming animals, where the visitors must remain in their cars while observing the animals. We believe that the user experience can be improved by developing a system that can automatically detect animals of interest, while the visitor is looking at them and provide visual information directly in the users field of view. The information available to the user could essentially be anything from general information about the type of animal, the name of the specific animal or what the animal is currently doing. To be able to augment information like this, it is necessary to first find out what the user is observing and then find and recognize any animals in the field of view. If the user was wearing a head-mounted-display, with cameras, the user could in theory observe the surroundings through the HMD unhindered, while the camera feed can be used for image processing. Using computer vision techniques, the animals in the field-of-view can then be detected and recognized. The goal of this project is to answer the question of how can we detect and recognize an animal and classify what action the animal is performing?

1.2 Related Work

The work in this report is build on a plethora of work in augmented reality, object recognition and action recognition. The previous work that has been done in these fields, do not map directly to our application as we work with a combination of constraints that are unusual for the above fields. The use of an HMD removes some of common constraints used in action recognition tasks, that relate to camera motion, while applying both detection and action recognition methods to animals in real time is very uncommon.

1.2.1 Augmented Reality

Augmented reality is a way of combining digitally generated data with a real world scene being observed. Augmented reality is often seen as 3D characters that are rendered into a real world scene, often interacting with a user or the scene itself. A similar example is through software sold with many web-cameras, that allow the user to digitally add a hat or mustache to a person observed by the camera.

For commercial use, the most commonly seen augmentation method is based on fiducial markers as implemented in the ARToolkit[6] which also supports 2D barcodes, natural features and custom patterns. Fiducial markers are a form of 2D barcodes that show a clear and unique pattern, which is known and can be detected by the AR software. The software then use the scale, rotation and placement of the marker to determine the size, placement and rotation of a 3D model to overlay the marker. Some AR implementations use simple features as known from the computer vision community, such as edge, point or corner detectors to augment graphics into scenes

without the use of classic markers. The main focus in augmented reality research has focused on markers based on markers with clear patterns or on human features such as eyes or faces. Our goal is to develop a system that can use the natural appearance and motion of animals as markers for augmentation purposes.

1.2.2 Object Recognition and Detection

For object detection and tracking, local descriptors such as the Scale Invariant Feature Transform[7] or Speeded Up Robust Features[8] are often used in a sparse implementation, using good features found in the image. The SIFT use Differences of Gaussians, image pyramids and orientation binning to gain scale and rotation invariance. The SURF method is to some degree inspired by SIFT, but use approximate Haar-wavelet responses as a part of the keypoint extraction stage. By using integral images, the haar-wavelet responses can be obtained with only 4 memory lookups for arbitrary kernel sizes, resulting in high computational performance.
The AdaBoost framework of Viola and Jones[9] was developed for face detection, but has since been applied to many other problems[10][11]. For human detection a more common approach is to use Histograms of Oriented Gradients, which is a dense descriptor based on image gradients and orientation binning. The HOG descriptor has been successfully applied to pedestrian detection problems and many forms[12][13][14]. A principally different approach is part based models, that use a probabilistic model of a human part composition. A collection of detectors are then used to detect the human parts and the composition is then matched against the probabilistic model[15][16][17]. The main focus in the field is on recognizing humans or more static scene objects such as number plates, cars or buildings. Some work has included horse detections, but animal recognition is a field not yet thoroughly explored. We seek to apply research done in the area of detection and recognition of humans to the problem of detecting and recognizing different animals, as a part of our system. The application requires real-time performance and must be able to detect non-rigid objects. The method should also integrate well with an action recognition step, which requires the method to be very generic and can not be constrained by shape assumptions.

1.2.3 Action Recognition

Since human motion is biological motion, the successful methods from human action recognition will mostly map very well to animal action recognition, as long as the human shape is not an important assumption of the method.
In action recognition there are two main methods of working with the data, global representations and local representations. In global representation the object of interest is extracted as a region of interest and the complete region is used in the recognition task[18]. Using silhouettes from background subtraction, Bobick and Davis[19] used motion histograms and motion energy images to represent motion. Efros et al.[20] used motion flow instead of silhouettes to avoid background subtraction, to compensate for motion due to camera movement they used tracking of the object. Thurau and Hlaváč[21] and Lu and Little[22] both use the grid-based histograms of oriented gradients representation for action detection. This method combines local and global features, as each cell in the grid is a local representation associated with the global ROI. Ikizler

et al.[23] combine the grid-based histogram of line segments with the optical flow method of Efros et al.[20] for human action recognition. A third method is to extract multi dimensional spacio-temporal volumes (STV) such as Jiang and Martin[24] who use shape flows over time, improving performance on cluttered backgrounds. By constructing 3D volumes, with time as the third dimension, the matching problem can be seen as a 3D template matching problem, instead of matching 2D shapes in different spaces. Yan et al.[25] extract minima and maxima from the STV, obtained from a multi camera view to create action sketches and project them to constructed 4D exemplars, using 3 spacial dimensions. The multi view setup help reduce sensitivity to noise and allow matching from arbitrary viewpoints.
The second main method is by using local representations. The local descriptors only detect certain points of interest on the object, by analyzing patches. By using patches the orientation of the object is of less importance for successful recognition, as long as the area matching the patch is visible. One way of choosing the patches are by looking at saliency. Laptev and Lindeberg[26] looks for patches with significant temporal and spacial variation using an extended version of the Harris corner detector. Willems et al.[27] uses integral videos to to efficiently calculate the determinant of a 3D Hessian matrix, which they use to define saliency.
Bregonzio et al.[28] use Gabor filtering to detect salient points, in an area of focus. In [29] Scovanner et al. extend the SIFT[7] descriptor to 3D. Wang et al.[30] found that a combination of image gradients and flow information generally results in better performance for local descriptors. Jhuang et al.[31] imitate the human visual system, based on biological research on the human brain, to build a system that works on both spacial and temporal data. In this project we aim to implement an action recognition algorithm that as seamlessly as possible integrate with an animal detection algorithm to build a framework combining the two to generate markers for augmentation purposes.

1.3 Contribution

In this project we develop a classification system that can be used in a head mounted display in a natural environment. The system detects and classify animals and their current action based on spacial contours and movement. We use HOG descriptors based on appearance data of frames from a video stream to detect and classify the animals. We then detect what action each animal is performing using HOG descriptors on motion frames of each detection region. We correct the motion frames with the mean motion vector between each frame to obtain a significant invariance to camera movement. We test the 'action recognition' stage with frame-to-frame motion maps and with multi-motion maps. Using multi-motion maps helps capturing cyclic motion and minimize errors from outliers during detection. The classifier is build as a cascaded SVM classifier, where the first stage detects and classify animals using single video frames. The second stage use the motion data to classify the action. The result from each cascade is appended in each detected region to build a history of what is seen in the region, which is then finally displayed on the screen along with a border around the animal.

PART I

ANALYSIS

Chapter 2

Platform

The system will consist of a Head-Mounted-Display, HMD, connected to a laptop, since the HMD does not have any processing capabilities. The HMD is a Vuzix 920AR model which features displaying 3D contents on the internal displays and recording with two front mounted stereo cameras. All inputs for the system is provided by the HMD, which is to be worn by the user of the system.

2.1 Hardware

The Vuzix 920AR has the following hardware:

- Two front mounted web cameras capturing up to 640x48030FPS
- Two built in displays for stereo- or mono-vision display in up to 1024x768
- 6-degrees of freedom motion sensor

  Pitch, Roll, Yaw, X-, Y- and Z-value

The HMD has two front mounted cameras that can be calibrated into a stereo camera setup. On the inside of the glasses are two displays that will allow the user to view any content provided by the system. The HMD also has a 6-degrees of freedom motion sensor that can provide rotation and translation data how the HMD is being moved. The motion sensor provides the data using 3-axes magneto-resistive sensors, 3-axes accelerometers and 3-axis gyros.

2.2 Interfaces

- VGA/DVI interface for display input
- USB2.0 connector shared by the 2 built in web cameras
- USB cable for power and tracker data

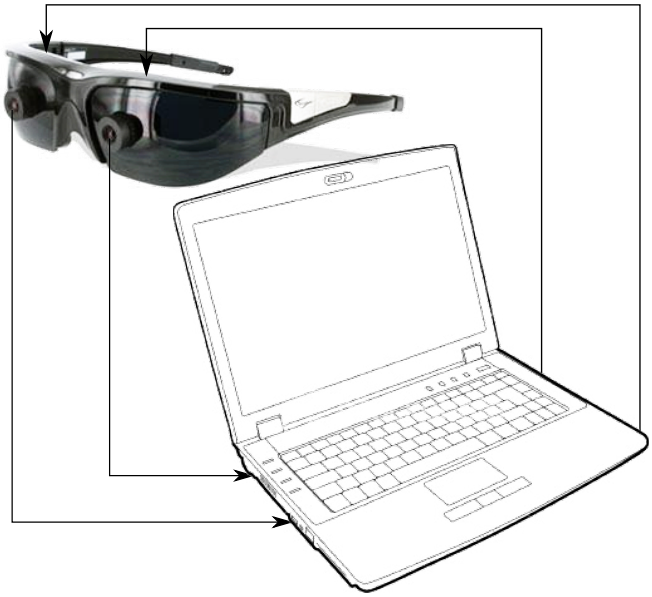


Figure 2.1: *The basic system diagram, showing the hardware relation.*

The idea for the prototype is to connect the HMD to a laptop that will compute the relevant data from the input sensors and display the result on the HMD displays. An illustration of the setup can be seen in fig. 2.1.

2.3 Features

The HMD has an additional set of features, that can be used either as the user or the developer for the platform. The main features of interest are:

- Side-by-side stereo display mode (SxS)
- Filtered 6 degrees of freedom sensor input
- Remote for On-Screen-Display for manual adjustments of the HMD

2.4 Extensions

With the rapid development of mobile platforms, such as smart phones and tablets, the HMD could very well be used with our system on such a platform instead of a laptop. This would greatly increase the mobility of the system and make for versatile deployment. A likely approach could be the development of a common framework, where a venue can easily build a plugin that seamlessly integrate with standard HMDs on all mobile platforms.

Chapter 3

Problem definition

3.1 Overall Requirements

The goal of this project is to develop an online system that is capable of detecting animals and their actions using a head mounted display, with cameras. From a user point of view the system must be able to do the following:

- Detect animals of interest in the Field Of View (FOV)
- Classify what type of animals are being observed and what they are doing
- Provide information about the animals in a fun and relevant way

The two first points are classification problem, while the last point is an Augmented Reality problem. This dictates a system structure resembling the one illustrated in fig. 3.1. The user requirements also allow us to define the technical requirements of the system to be implemented.

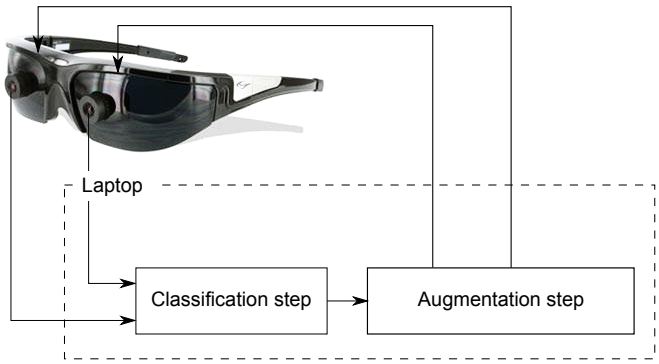


Figure 3.1: *A simple system outline, defining the blocks that should be implemented in the system running on the laptop.*

3.2 System Requirements

3.2.1 Classification step

To achieve the goal as described above, the classification step will follow a common structure for classification problems in computer vision. The structure has a feature extraction step where features will be extracted from the input images. The features are then encoded in a descriptor step, which generates a common representation for one or more features, to make them comparable in the final step, the classifier. The classifier step uses the descriptors to determine if the observed region is a known class and which. The structure is illustrated in fig. 3.2.

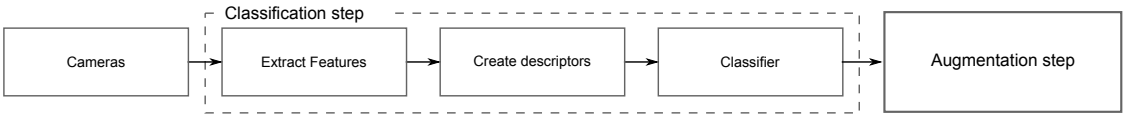


Figure 3.2: *The structure of the classification step.*

Classifier

- Detect and classify different animals
- Classify which action is being performed

In order for the classifier to distiungish between positive data from negative data, the data needs to be vectorized into a descriptor.

Descriptor

A descriptor is a container that is used to analyze and store the description of a feature found in the input data. The data describing the classes we are interested in is obtained from the front mounted cameras of the HMD, which means the data available is video and images. The videos are essentially lists of images which are subsequent to each other in a temporal domain. The descriptor should therefore be chosen among descriptors that live up to the following:

- Good performance on image data
- Good tolerance to noise
- Scale invariant
- Intensity invariant

To use a descriptor it is necessary to chose a set of features that will take advantage of the descriptors strengths.

Features

The features are some kind of descriptive data obtained from the input. This can be the input itself or a transformation of the input. In image and video data, features such as points, edges or motion are among the common features. We want to choose our features so that they comply to the following:

- Data representation as an image
- Portray structures representative to the animal type
- Portray structures representative to the action of the animal
- Invariance to camera movement

We choose to represent the features as images, since our main source of data is image data. This choice will likely enable the use of only a single descriptor for the classification step. Invariance to camera movement means that the when the subject is observed in the feature space, it should appear the same no matter if the camera is still or moving. This is important since the platform is a HMD and the user can not be expected to be perfectly still when detection and classification is being performed.

3.2.2 Augmentation step

The goal of augmentation step is interpret the results from the classifier and utilize the available means of communicating with the user to augment useful information. It must be designed to comply with the following:

- Extract information from the classification step
- Interpret the results
- Generate useful content to augment
- Augment the information in a convenient way to the user

3.3 System Limitations

The platform of the system limits the application in several ways. One major limitation is to the real time requirement, as the HMD should be connected to a computer during use. In this project we focus on implementing the system on a laptop. The laptop has limited computation power over a desktop computer or computer cluster. The laptop will in a user scenario be running on battery power, which will further limit the computation power, but also mean that excess processing will result in low battery time. A second set of limitations come from the HMD, which will provide the data for our system and be the target of our output. This means that limitations on the HMD will directly affect detection performance and the experience of the

system. The HMD has two built-in cameras, which are each able to provide video streams in a resolution of 640x480 pixels at 30 FPS in the YUY2[32] color format. Since both cameras are connected to an internal USB2.0 HUB and connected to the PC with one common USB cable, the maximum combined data rate is 480 Mbps[33, p. 46 table 5-5]. The data rate of each camera at 640x480 at 30 FPS YUY2 is:

$$datarate = 640 \cdot 480 \cdot 2.5 \cdot 30 \quad Bps \tag{3.1}$$
$$= 23.04 \quad MBps \tag{3.2}$$
$$= 184.37 \quad Mbps \tag{3.3}$$

Communications on the USB2.0 bus will at it's fastest configuration, High-speed Isochronous Transaction with a data payload of 1024 bytes[33, p. 46 table 5-5], be able to transfer useful data at:

$$57344000\,Bps = 57.344 \quad MBps \tag{3.4}$$
$$= 458.753 \quad Mbps \tag{3.5}$$

This calculation does not take into account any overhead from data encapsulation, bit stuffing or time slot negotiations on the USB2.0 hub in the HMD. With a combined rate of the two cameras of 368.64 Mbps raw data through at least one USB2.0 hub, the HMD is likely to approach the limits of the USB2.0 connection. Empirical tests showed that our test computers were only able to capture at approximately 26 FPS from the two cameras simultaneously. To avoid fluctuating frame rates, we chose to lock the frame rate at 20 FPS.

The screens of the HMD both operate at a maximum resolution of 1024x768 in the mono-vision setup, meaning that the image displayed on both screens is the same. When switching the HMD into stereo mode, the screens are set up in a Side-by-Side, SxS, stereo mode, meaning that the each screen show half of the image send on the VGA connection, left screen shows the left half, right screen the right half. In this mode the resolution stays the same, meaning that the horizontal resolution of each screen is effectively halved and stretched when the HMD is in SxS mode.

According to the specification, the HMD provides a diagonal field of view of 31°, which is a very narrow field of view, as humans rely heavily on their peripheral vision when navigating. The company producing the HMD is currently working on VR glasses in wide screen and with higher resolution, which may address some of these problems, but none of the product announced are equipped with cameras yet. Though the limitations in resolution and field of view is essential for the success of a final product, these limitations are of no importance to a proof of concept, and will therefore not be further addressed in this report.

3.4 Scope

3.4.1 Classification stage

The classifier stage of the system must live up to the following requirements:

- Detect any animal seen on the screen of the HMD
- Classify what animal is being observed
- Classify which action is being performed by detected animals
- Works during low to medium camera rotation speed
- Works in most lighting conditions
- Compute new result at least every 50 ms

Low to medium camera rotation is defined as the shaking and rotation of the camera when the user is observing a mostly still subject (low rotation speed) or when the user is tracking a subject that is moving (medium rotation speed). The system is not required to detect and classify during high rotation speeds, defined as when the user is navigating or rapidly shifting attention. We choose to implement this invariance for the pitch and yaw rotations of the camera only as camera roll is not expected to be a common rotation type when observing an animal. The lighting conditions the system should work in, is defined as daylight and indoor lighting where the subjects are clearly visible in the images. As this report is a proof-of-concept, the computational speed of the algorithms are of less importance, as there are many ways to improve on this, but it is not the main focus in this report. The amount of animals and actions the system will be trained on is limited since the data must be recorded using the HMD and the data set is hard to obtain. The specific animals and actions trained in this project can be seen in part III.

3.4.2 Augmentation

In this project we will focus on the design and implementation of the classification stage. The augmentation step will therefore only be implemented as a basic overlay, but could in principle be extended to elaborate visual or audible augmentations. The requirements for the augmentation step will be:

- Mark animals of interest
- Label the markings with information on what type of animal is being observed
- Label the marking with the action the animal is currently performing
- Display new markings and labels on every classification update

Chapter 4

System Definition

4.1 Classification step

4.1.1 Choice of classifier

The classifier needs to detect and classify animals and their actions and have the ability to perform these online. Boosted classifiers and SVMs, have proven effective in detecting humans, human features and other static objects[34][35][12][36].

Boosted classifiers are often used in combination with simple descriptors, such as Gabor or Haar kernel responses, to train a strong classifier from an array of weak classifiers based on these descriptors. The weak classifier is trained by folding positive and negative training samples with the kernels, using the responses to define a threshold for the weak classifier. The responses of the positives and negatives are then used to find the combination of weak classifiers that separate the positive samples from the negative samples the best. By cascading such classifiers, keeping the number of weak classifiers low on the first levels of the cascade, the computation time of a boosted classifier can be reduced to a point where online performance can be achieved. The downside to the boosted classifier is that it is very sensitive to inaccurate training data and is prone to over fitting on small training sets. To avoid these problems, the classifier must be trained with large amounts of both positive and negative training samples to get good performance.

SVM classifiers use descriptor vectors of positive and negative samples, mapped into a high dimensional space to find the hyperplane best separating the positives from the negatives. The linear SVM classifier maps the descriptor vectors directly into the high dimensional space and finds the optimal hyperplane separating the samples, which is in most cases a sufficient method. If the samples are not linearly separable a kernel based SVM classifier can be used. This method folds the descriptor vector with a kernel, such as a Gaussian, to map the descriptor into a, typically higher dimensional, Hilbert space where the samples become linearly separable. Since the descriptor vectors used in an SVM are usually very high dimensional, the folding operation in kernel based SVM classifiers makes this method slower than a simple linear SVM.

The boosted classifier is prone to shaping the separation function to a small positives set, as it tries

to find a way to detect all positives and negatives perfectly. This can have a negative effect, when the boosting algorithm train a classifier that works very well on the small training set, it may not generalize well. The linear SVM classifier will on the other hand try to find the hyperplane that separates the positives from the negatives with a gap wide as possible. Since the hyperplane is linear, it does not take shape of the training set and might in many cases be more invariant to over fitting. SVM classifiers have also proven to perform very well on pedestrian recognition[12] and human detection using appearance and optical flow[37].

In this project we will be using footage of animals performing different actions, which must be recorded using the HMD provided. We must therefore record all the training data with the HMD and cannot use existing datasets, which means our dataset will be quite limited. Considering that the SVM classifier is well suited for small datasets and has already been proven to work well with detection of humans, we have chosen to use the SVM classifier as the classifier step in this project.

4.1.2 Choice of descriptor

In the previous section the features for the detector were chosen so that all features can be described as images, thus allowing the use of one descriptor optimized for image analysis. Most current object detectors use variations of either Scale Invariant Feature Transform[7] or Histograms of Oriented Gradients[12] as the descriptor.

The Scale Invariant Feature Transform, SIFT, descriptor is a sparse descriptor that uses pyramid scales of Differences of Gaussians. The descriptor is designed to find unique and easily recognizable key points in images, which are well suited for detection and recognition of objects. The SIFT algorithm uses pyramidal scaling, also known as octaves, to become scale invariant. The SIFT descriptor is often used for object recognition, sparse optical flow and for tracking objects. The SIFT descriptor is computationally a very efficient algorithm and many real time implementations already exist. Since the SIFT descriptor is sparse, it typically uses collections of key points, but does not store spacial relation between features. The spacial relation between key points may very well contain important information about the pose of the animal and thereby what action is being performed. To be able to determine action from motion, it is important that the key points are placed on limbs that move with high correlation to the action being performed. This can not be guaranteed by the SIFT descriptor as the best keypoints may not necessarily be those best separating one action from another.

The Histograms of Oriented Gradients, HOG, is a dense descriptor, which uses every pixel in the input image to create its descriptors. Each descriptor is composed by values optained from histograms of the gradients in the image. Each histogram is found by dividing the gradient orientation into bins and adding the gradients amplitude to their respective bins. The image is divided into blocks, which is again divided into cells from each a histogram is made. The histograms are normalized with respect to the other cells within the same block. The HOG descriptor is very good for coding spacial form, which is very important for action recognition, since movement as well as actual pose contains valuable clues to which action is being performed. Since the HOG descriptor is a dense descriptor which uses a range of image processors, the algorithm is computationally heavy and the obtainable frame rate is low. According to Dalal &

Triggs[12] the computational performance of the HOG descriptor in a linear SVM classifier was under a second per frame in a 320x240 image with 4000 detection windows, which must mean that each frame takes more than 500 ms to process.

The SIFT algorithm is by far the most computational efficient of the two algorithms, but the lack of spacial relation of the features and the fact that there is no way to know if the best features also are useful for describing movement, makes the SIFT descriptor a less attractive choice. The HOG descriptor on the other hand, encodes features for the spacial shape of the object and has in other papers been proven useful for both spacial and temporal matching. The downside being that it is a very computational heavy descriptor. We choose to use the HOG descriptor in this project, since the real time requirement is of less importance than the classification performance requirements.

4.1.3 Choice of features

According to the classification step requirements, we are interested in detecting which animal is being observed and what action is being performed. To do this efficiently, it is necessary to find and extract features that are good for each of these tasks.

Appearance

To detect an animal and classify which type of animal is being observed, the appearance of the animal contains many valuable cues. As humans we can often tell what basic class of animal we are looking at, just by observing it's silhouette. This indicates that the shape of the animal itself, may contain valuable information. For human detection, the unaltered input image from cameras has already proven to perform well when used in combination with the HOG descriptor[12][14]. Combining the input image with optical flow in the HOG descriptor seem to further improve detection performance[37]. We therefore choose to use the input image as a feature for our animal classifier and end up testing if a combination of the input image and the optical flow might improve the performance.

Motion

To detect what action the detected animal is performing, it seems natural to look at the motions of the animal. The work of Ikizler et al.[23] show that the use of optical flow is useful for human action detection. We decide to use the CUDA implementation[38] of the optical flow algorithm developed by Werlberg et al.[39] to generate flow maps. The flow maps of this method are very detailed and shows detailed silhouettes of the moving objects, which implicitly encodes poses as well as movement. The optical flow algorithm extracts motion between two frames, providing frame-to-frame motion. Using frame-to-frame motion as a feature may very well encode very fine movements for the detector, but many actions such as walking exhibit cyclic motion patterns. The classification performance of cyclic motions might improve when running classification on data collected over multiple motion frames. To test if the system benefits from using multiple motion frames, we implement a simple multi-temporal-motion-model. The model uses motion

over several frames to encode the relation between frame-to-frame motions and improve encoding of cyclic motions.

Since appearance images might provide information about a specific action, one could imagine features as a combination of appearance and motion frames. On that behalf, we therefor choose to test motion only features and a combination of appearance and motion features.

4.1.4 Summary

The final classifier will be an SVM classifier using HOG descriptors to detect what animal is being observed and what kind of action is performed using the appearance images and / or motion frames. Motion frames will be extracted from the current appearance image and the previous one. An illustration of the chosen classification step structure can be seen in fig. 4.1.

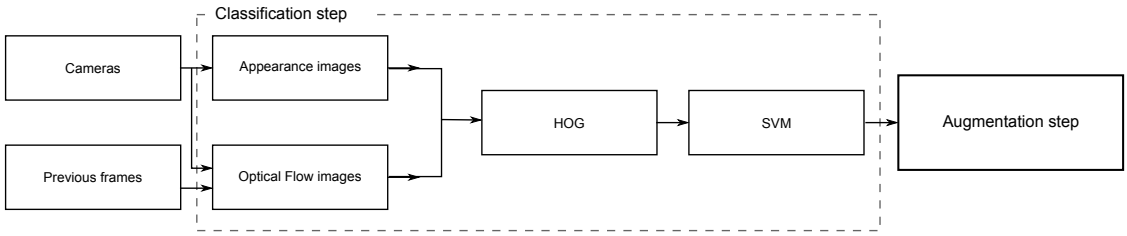


Figure 4.1: *The chosen structure of the classification step with the choice of components.*

4.2 Augmentation step

The augmentation step will be a very simple augmentation, as the focus of the report is how to obtain data that is useful for augmenting information in the context defined in chapter 1.1. The augmentation step will receive a region, marking the location and extend of the detected animal. The classification step also provides class labels containing information on what animal and what action has been performed. This information can be used to augment information in many ways, but since this is not the focus of this project, we chose to augment the detection information as described in section 3.4 and illustrated in fig. 4.2.

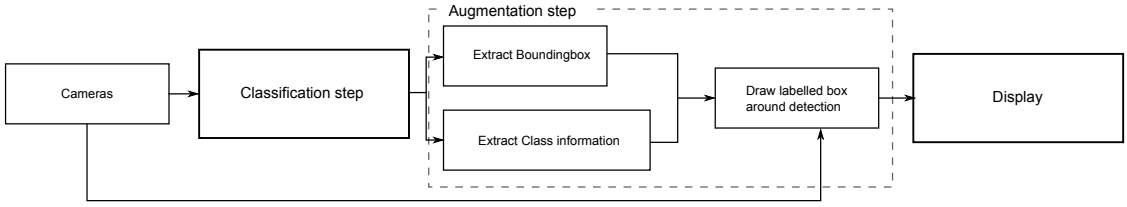


Figure 4.2: *The functionality structure of the augmentation step.*

Marking the bounding box

The classification step outputs a bounding box of the detection. The coordinates of this bounding box is converted to a rectangle and rendered onto the output frame.

Labeling the bounding box

The classification step also provides a history of classes or a combined class for the bounding box. When a history of classes are present, a string is appended to generate a meaningful string, describing the animal and action. The text is then rendered onto the output frame on top of the bounding box.

Displaying the result

We decide to render the resulting frame to the screen using OpenGL[40] to improve performance and reduce lag. The output frames, overlayed or not, are copied to an OpenGL texture on the graphics card, when all other operations have finished. The texture is applied to a quad and rendered to a window. Since the HMD uses SxS stereo rendering, we implement support for stereo images also. The left and right frame is rendered to a quad each, each filling the left and right half of the window respectively. When viewed in fullscreen, this render method will correctly display the images on each of the screens in the HMD.

Summary

The final augmentation step will take the detected regions from the classification step and an image, to augment the data into, as inputs. From the detected regions a class history will be used to generate descriptive text. The text and the bounding box of the detected region will be rendered on top of the input image and into an output frame. The output frame will then be rendered as textures on quads in an OpenGL window for display on the HMD. The final structure can been seen in fig. 4.3. The specific implementation of the augmentation step will not be discussed further in this report.

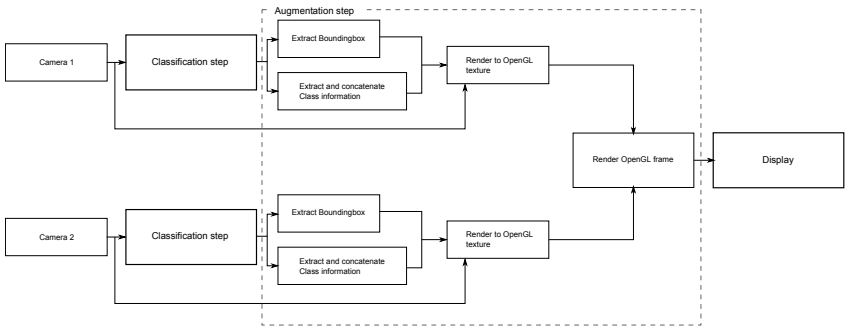


Figure 4.3: *The final implementation structure of the augmentation step.*

PART II

PROPOSED SOLUTION

Chapter 5

The classification step

As has been found in chapter 4, a classification step needs to be developed in order to find the regions containing an animal and the action these animals are performing. The classifier needs to be able to classify the animal and its actions based on appearance- and motion-data, where the appearance data consists of plain RGB frames and the motion data consists of motion information generated by one or several motion frames created by the optical flow algorithm. A simple and intuitive way of creating a classification step, would roughly be the one illustrated by the blockdiagram in figure 5.1.

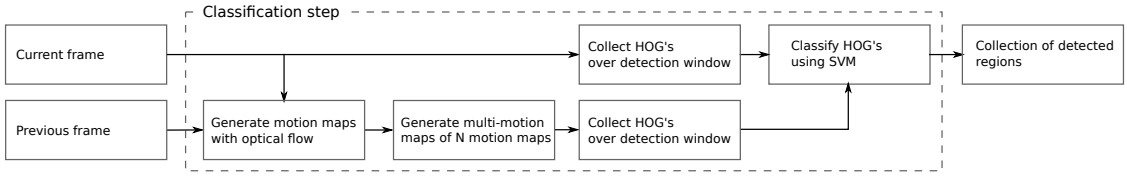


Figure 5.1: *Classifier step using optical flow and HOG combined with a SVM classifier. The classification step returns the regions containing an animal and the action performed by this animal. The multi-motion maps used by HOG is generated by the N previous motion maps, which is created by the optical flow algorithm.*

Note that the detection window is defined as the area from which the HOG descriptors are generated and the detected region is found as the output area generated from running non-maximum suppression on the positive classified detection windows. The motion frames are from now on called motion maps or multi-motion maps, since the motion maps can consist of information from one or several motion frames respectively.

The classification step in figure 5.1 would however not be particular efficient, which leads to the approach of using a cascade of classifiers instead.

5.1 Cascade of classifiers

In order to decrease the computation time of computing all the possible class and action combinations in one step, a well known method of arranging the classifiers in a cascade of classifiers will be used. The cascade of classifiers consists of different layers of classifiers arranged as a decision tree. In each layer or stage of the cascade, a decision is made if the detection window contains the class. An example of such a cascade of classifiers is illustrated in figure 5.2.

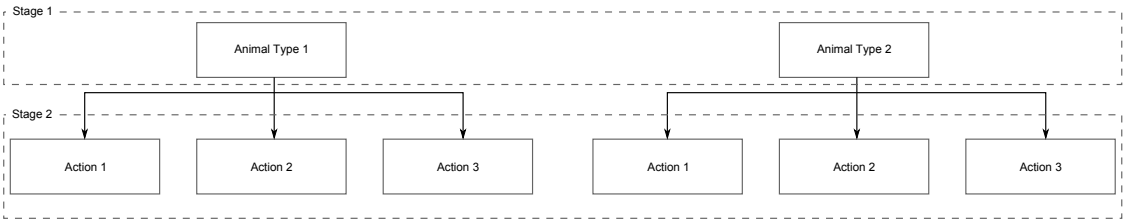


Figure 5.2: *Cascade of classifiers tree, where the type of animal is found in the first stage and the action for the respective animal in the second stage. The regions found to contain the animal in the first stage, will be used to generate new detection windows for the action classifiers.*

The purpose of the cascade of classifiers is to use only a few classifiers in the first stage, to eliminate large areas of the frame that does not contain the searched classes. In the following stages of the cascade fewer regions will be candidates of classification, and as a natural consequence of this, the candidates can be classified by more specialized classifiers. In the example illustrated in figure 5.2, the classifier only needs to search for two types of animals in the whole frame in stage 1. This leaves only the regions in which the animal is detected for the next stage. Detecting the action performed by the animal in stage 2 is therefore much less of a challenge both computational-wise and detection performance-wise. In comparison, if no cascade was used, the classifier would have to compute six different classifiers in the whole image, which is a harder challenge both computational-wise and detection performance-wise. Detection performance-wise since the classifiers can not be specialized in either appearance or action recognition, computationally since the amount of descriptors to be calculated for the image is greatly increased.

The cascade of classifiers can be extended to an arbitrary number of stages and number of classes in each stage. We choose a two stage setup as the one shown in fig. 5.2, i.e. stage 1 is detecting the regions containing a specific animal and stage 2 its actions. Since different types of animals might have different types of actions with different compositions, it is sensible to identify the animal first, before detecting which action is being performed.
From this point on stage 1 will be referred to as the 'animal detection and recognition' stage and stage 2 as the 'action recognition' stage.

Incorporating the cascade approach in the classification step, yields the final overall classifier step illustrated in figure 5.3.

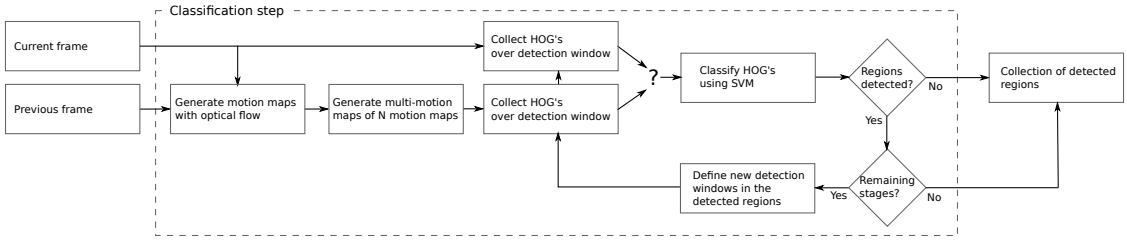


Figure 5.3: *The classifier step rearranged as a cascade. Here the question mark ? is indicates a choice of using either HOGs encoded from the appearance frames, motion data or a combination of the two, for the current stage in the cascade. New detection windows are generated from all the detected regions in the current stage and, if anymore stages remain, they are passed on.*

Two things should be noted when using the above classification step,

1. The question mark in the block diagram illustrates the choice of using any combination of the HOG descriptors, in order to yield the best detection performance with respect to the animal and action classification.

2. In any of the stages of the classifier, if no regions are detected, the already detected regions from previous stages are collected as detection results. This serves the purpose of detecting e.g. an elephant even though its actions was not recognized.

The ability to classify e.g. an elephant even though the action was not recognized, is important with respect to robustness. The robustness comes from the ability to keep track of where the elephant is located, even though the action is not recognized. The importance of keeping track of where the elephant is located becomes evident in section 5.2.6. From a user point-of-view it is also important to detect the animal, even though the action is not recognized, as missing detections are worse than less detailed detections.

The other ability to choose the combination of the HOG descriptors individually, gives an advantage with respect to detection performance. It is reasonable to assume that the appearance HOGs are better at classifying the type of animal than using only the motion data HOGs. Likewise, the motion data HOGs are presumably better at classifying the animals action than the appearance HOGs. At last, a combination of both the appearance and the motion data HOGs might yield better results, which was the case in Dalal & Triggs[37], for both the animal and action classification. The different combinations of HOGs will be tried and verified for each stage in the cascade of classifiers in part III.

In the following, each block in the blockdiagram 5.3 will be explained in the following order:

Generate motion maps with optical flow:
The generation of motion maps with the help of a dense optical flow algorithm, is explained in section 5.2.
Generate multi-motion maps of the last *N* motion maps:
By utilizing the last *N* found motion maps, one can generate multi-motion maps, which is explained in subsection 5.2.5.

Collect HOGs over detection window:
How to encode appearance or motion maps into HOG descriptors are covered in section 5.3.
Classify HOGs using SVM:
The training and classification using SVM, is clarified in section 5.4.

5.2 Motion maps

It was found in chapter 4.1.3, that the implementation of multi-motion maps could possibly improve the action detection step when using the optical flow algorithm. The improvement comes from the use of movement over frame sequences in a video instead of only frame-to-frame movement.

Let the single frame vector motion map be defined as M, which consists of the dense pixel movement (optical flow) between two consecutive frames. Let the x- and y-component of the vector motion map be defined as the motion maps M_x and M_y respectively. Then let the vector multi-motion map found by evaluating several motion vector maps be defined as $\bar{M}$ and its components as $\bar{M}_x$ and $\bar{M}_y$.

Since the resulting vector multi-motion map $\bar{M}$ heavily depends on the implementation of the vector motion map M, the following section will clarify how to estimate the later. Roughly three problems arises in estimating the motion map components M_x and M_y:

Firstly, how do one estimate and measure motion between two frames? There has long existed good algorithms to approximate optical flows, from which implementations exists in various optical flow libraries free to download and use. Since the main aim for this project is not to develop an optical flow algorithm, a CUDA implementation of an optical flow algorithm is used [38]. The algorithm [39] proposed by Manuel Werlberger et al. used in this library is explained in section 5.2.1.

Secondly, how do one find the relative motion between the camera and the scene, making the resulting M_x and M_y motion maps robust to camera movements like translation and rotation. A method solving this, is explained in section 5.2.3.

Thirdly, the generated motion maps need to be cropped to extract the objects of interest in the generation of training images. In order to do this, the motion maps need to be saved to a RGB 24-bit format. Since the motion maps will consist of both negative and positive floating point values, they need undergo a transformation to save all the motion information correctly, see chapter 5.2.4.

In order to show how the flowlib library differentiates from other and more classic approaches, it has been chosen to first explain the method proposed by Horn and Schunck [41], from which many state of the art optical flow algorithms originate.

5.2.1 General optical flow

In general, and also done by the algorithm proposed by Horn and Schunck, optical flow algorithms estimate flow under the assumption that patterns in an image will not change brightness when moving, i.e. the image brightness at a point (x,y) in the image at time t, denoted by $E(x,y,t)$, will have the following constraint:

$$\frac{dE}{dt} = 0.$$

Using the chain-rule for differentiation gives

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0.$$

The velocity can now be defined as the vectors

$$u = \frac{dx}{dt} \quad \text{and} \quad v = \frac{dy}{dt}$$

This gives one equation with the two unknows u and v

$$E_x u + E_y v + E_t = 0, \tag{5.1}$$

The equation can not be solved without introducing addition constraints and is commonly known as the aperture problem. The above constant brightness equation constrains the optical flow algorithm, see figure 5.4.

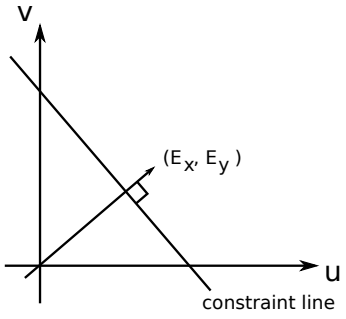


Figure 5.4: *The brightness equation 5.1, constrains the velocity (u,v) to lie on a certain line perpendicular to the brightness gradient vector (E_x,E_y) in velocity space.*

The aperture problem

If every pixels could move independently, it would be impossible to estimate the optical flow, due to the aperture problem. To solve this problem Horn and Schunck view opaque objects as objects with finite size and undergoing rigid motion or deformation. This assumption implies that neighboring points have similar velocity and the brightness patterns varies smoothly almost everywhere. This constraint is called the smoothness constraint and is defined as the minimization

of the sum of the squares of the laplacians of the x- and y-component of the flow.

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad \text{and} \quad \nabla^2 v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}$$

Estimation the partial derivaties of brightness change and the Laplacian equations

In order to solve the two equations for the two variables u and v, Horn and Schunck estimated the partial derivatives of the brightness change and the laplacians of the x- and y-component of the flow. The partial derivaties of brightness change are estimated as:

$$E_x \approx \frac{1}{4}\left\{E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i+1,j,k+1}\right\}$$
$$E_y \approx \frac{1}{4}\left\{E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k} + E_{i+1,j,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i,j+1,k+1}\right\}$$
$$E_t \approx \frac{1}{4}\left\{E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j+1,k} + E_{i+1,j+1,k+1} - E_{i+1,j+1,k}\right\}$$

The Laplacians of the velocity of the flow are estimated as:

$$\nabla^2 u \approx 3(\bar{u}_{i,j,k} - u_{i,j,k}) \quad \text{and} \quad \nabla^2 v \approx 3(\bar{v}_{i,j,k} - v_{i,j,k})$$

Where the local averages are defined as follows

$$\bar{u}_{i,j,k} = \frac{1}{6}\left\{u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\right\} + \frac{1}{12}\left\{u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}\right\}$$
$$\bar{v}_{i,j,k} = \frac{1}{6}\left\{v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\right\} + \frac{1}{12}\left\{v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\right\}$$

These estimations can now be used to solve the velocity of the optical flow u and v.

Solving the the constant brightness and the smoothness constraint

To solve the two constraints, Horn and Schunck defined the problem as a minimization problem given the equation for the rate of change in brightness:

$$\delta_b = E_x u + E_y v + E_t$$

and the departure from the smoothness in the velocity flow

$$\delta_c = (\bar{u} - u)^2 - (\bar{v} - v)^2.$$

By introducing a weighting factor a^2, which is introduced due to the fact that the brightness measurements will be corrupted by quantization error and noise, so that δ_b can not be expected to be equal zero, the total error which has to be minimized is

$$\delta^2 = a^2 \delta_c^2 + \delta_b^2$$

By differentiating δ^2,

$$\frac{\partial \delta^2}{\partial u} = -2a^2(\bar{u}-u) + 2(E_x u + E_y v + E_t)E_x$$
$$\frac{\partial \delta^2}{\partial v} = -2a^2(\bar{v}-v) + 2(E_x u + E_y v + E_t)E_y$$

yields two equations, which can be set equal zero in order to find u and v. This yields the two new equations

$$(a^2 + E_x^2)u + E_x E_y v = (a^2\bar{u} - E_x E_t)$$
$$E_x E_y u + (a^2 + E_y^2)v = (a^2\bar{v} - E_y E_t),$$

which can be arranged in a in the form

$$(a^2 + E_x^2 + E_y^2)(u - \bar{u}) = -E_x[E_x\bar{u} - E_y\bar{v} + E_t]$$
$$(a^2 + E_x^2 + E_y^2)(v - \bar{v}) = -E_y[E_x\bar{u} - E_y\bar{v} + E_t],$$

where Horn and Schunck defines a^2 to be roughly equal to the expected noise in the estimate $E_x^2 + E_y^2$.

With these equations, one is able to estimate the optical flow velocities u and v, which is illustrated in figure 5.5. In general, the Horn and Schunck method and variations from it, suffers from

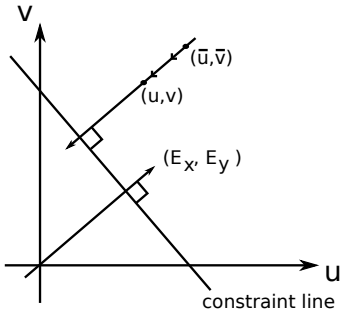


Figure 5.5: *The values of the optical flow velocities u and v which minimizes the error, lies on a line drawn from the local average of the flow velocity perpendicular to the constraint line.*

the three major problems, see [39]:

- **Untextured regions**: In poorly textured regions, no matching can be performed and hence strong regularization is needed to propagate the motion from textured regions into the untextured areas.
- **Occlusions**: As soon as motion is present in the images, regions are occluded or exposed by moving objects. Those occluded regions cannot be matched between the image frames.
- **Small scale structures**: Small scale structures with a relative large movement are hard to preserve in the motion field sine isotropic regularization does not adapt well to the local image structures.

In order to cope with these issues, it has been chosen to use a robust and popular library called flowlib [38], which uses an approach called Motion estimation using Non-Local Total Variantion Requlariza tion [39]. This approach tries to solve the above mentioned issues.

5.2.2 Motion estimation using Non-Local Total Variantion Requlariza tion

The following approach by Werlberger et al. is somewhat similar to the one proposed by Horn and Schunck, i.e. they are trying to minimize a data term and a regularization term, where the data term is a measure of the similarity of the input images for a given motion field and the regularization term ensures motion field smoothness. The data term would in the Horn and Schunck method be the constant brightness term and the regularization would be the velocity smoothness terms given by the laplacians of (u,v).

A variational motion estimation model is typically given by the minimization problem

$$\min_u \{R(u) + D(I_0, I_1, u)\},$$

where the unknown motion field $u = (u_1, u_2)^T : \Omega \to \mathbb{R}^2$ and the images I_0 and I_1 are all defined in the image domain $\Omega \subset \mathbb{R}^2$. Note that the variable u has been redefined to keep the authors notation. $R(u)$ is the regularization term and $D(I_0, I_1, u)$ is the data term. In the case of Horn and Schunck, the regularization and data term would be

$$\begin{aligned} R(u) &= a^2 \delta_c^2 \\ D(I_0, I_1, u) &= \delta_b^2. \end{aligned}$$

Note the slightly different notation used in this approach. Instead of referring to the brightness E in the images, $I(x)$ is now a reference to the pixel value in image I at coordinate $x = (x_1, x_2)$.

Second Order Approximation of the Data Term

As was the case in the Horn and Schunck method and a lot of other approaches of estimating the optical flow, the data term is based on the constant brightness constraint. If the illumination in the images changes, this constant brightness assumption would fail. To avoid this, a lot of approaches pre-process the images in order to eliminate or reduce the influence of illumination changes. This approach by Manuel Werlberger et al. reduce the influence of illumination by using the normalized cross correlation as a similarity measure between the two images. The normalized cross correlation is invariant to multiplicative illumination changes.

Let $\delta(x, (u(x_1), v(x_2)))$ be the pointwise data term depicting the truncated normalized cross cor-

relation. This gives the following data term

$$D(I_0, I_1, u) = \lambda \int_\Omega \delta(x, u(x))dx$$

To calculate the cross correlation, one must first find the mean and the variance. The mean of a certain patch is found as follows

$$\mu_0(x) = \int_\Omega I_0(y) B_\Sigma(x-y)dy \quad \text{and}$$
$$\mu_1(x) = \int_\Omega \tilde{I}_1(y) B_\Sigma(x-y)dy,$$

where B_Σ denotes a box filter of width $\sum$ and $\int B(z)dz = 1$ and $\tilde{I}_1(y) = I_1(y + (u_0))$ the warped image. The standard deviation $\sigma(x)$ is found as

$$\sigma_{0,1}(x) = \sqrt{\int_\Omega (I(y) - \mu(x))^2 B_\Sigma(x-y)dy}.$$

The normalized cross correlation can then be found as

$$NCC(x, u(x)) = \frac{1}{\sigma_0(x)\sigma_1(x)} \cdot \int_\Omega (I_0(y) - \mu_0(x))(\tilde{I}_1(y) - \mu_1(x)) B_\Sigma(x-y)dy.$$

Note that in order to account for local distortions within the correlation window, Werlberger et al. uses the warped image $\tilde{I}_1(x) = I_1(x + u_0)$ in the matching term. Also to be more robust against occlusions, they define the truncated normalized cross correlation as follows

$$\delta(x, u(x)) = \min\{1, 1 - NCC(x, u(x))\},$$

which discards all negative correlations.
This is a non-linear function, which can not easily be solved with numerical methods. In order to linearize the term, a direct second-order Taylor expansion is performed, yielding

$$\delta(x, u(x)) \approx \delta(x, u_0(x)) + (\nabla\delta(x, u_0(x)))^T (u(x) - u_0(x)) + (u(x) - u_0(x))^T (\nabla^2\delta(x, u_0(x)))(u(x) - u_0(x)), \tag{5.2}$$

where $\nabla^2\delta(x, u_0(x))$ denotes a semi-definite approximation of the Hessian matrix

$$\nabla^2\delta = \begin{bmatrix} (\delta_{xx}(x, u_0(x)))^+ & 0 \\ 0 & (\delta_{yy}(x, u_0(x)))^+ \end{bmatrix}$$

To summarize, the data term $D(I_0, I_1, u)$ is now expressed as a Taylor expansion of the normalized cross correlation instead of the more traditional constant brightness term, which will make the matching more robust against illumination.

Non-Local Total Variation Regularization

As was the case with the approach by Horn and Schunck, a second equation was needed in order to solve the velocity of the optical flow. In this approach, the second equation is called non-local

total variation (NL-TV) and utilizes the coherence of neighboring pixels kinetic behavior. As many other optical flow regularization algorithms, it is based on the belief that objects will move in a similar motion pattern.

NL-TV incorporates this information directly, yielding

$$R(u) = \int_\Omega \int_\Omega w(x,y)(|u_1(x) - u_1(y)|_\varepsilon + |u_2(x) - u_2(y)|_\varepsilon) dy dx, \tag{5.3}$$

where $|q_\varepsilon|$ denotes the Huber norm given by

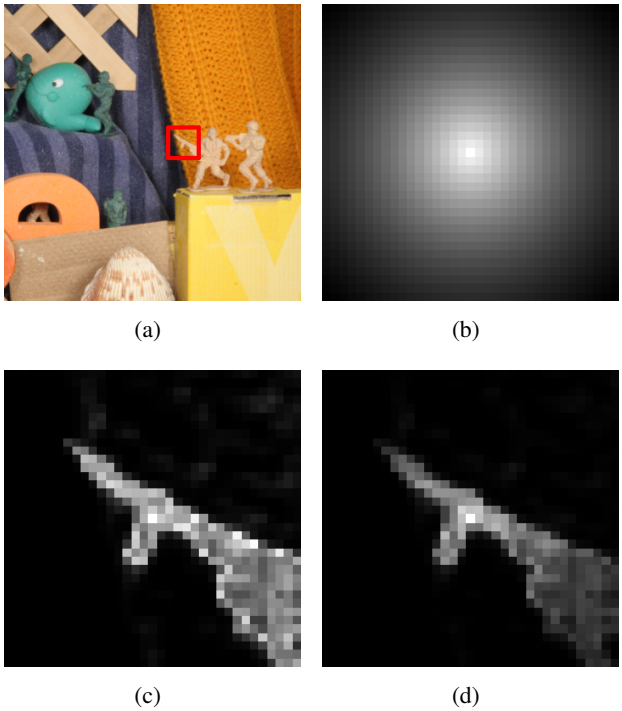
$$|q|_\varepsilon = \begin{cases} \frac{|q|^2}{2\varepsilon} & |q| \leq \varepsilon \\ |q| - \frac{\varepsilon}{2} & \text{else} \end{cases}$$

The Huber norm is used instead of the total variation in order to avoid piecewise constant motion fields. The function $w(x,y)$ is a weight function used to define the mutual support between the pixel at position x and y. The function is defined as follows

$$w(x,y) = e^{-\left(\frac{\Delta_c(x,y)}{\alpha} + \frac{\Delta_s(x,y)}{\beta}\right)}.$$

$\Delta_c(x,y)$ denotes the color similarity between $I_0(x)$ and $I_0(x)$ as the Euclidean distance in the Lab color space and $\Delta_s(x,y)$ is the Euclidean distance in the spacial domain between x and y.

Figure illustrates how the weighting function $w(x,y)$ has an impact on an image patch. As can



(a) (b)

(c) (d)

Figure 5.6: *Euclidean distance weights (b), color similarity weights (c) and the combined weights (d) for the subregion marked in (a). These pictures have been borrowed from [39]*

be seen in figure 5.6(d), the function $w(x,y)$ will encode a low level segmentation in the image. This will help the regularization term to only weight similar motion fields within the same segmentation.

Estimating the motion field

In order to estimate the motion field u, one has to solve the minimization

$$\min_u \{R(u) + D(I_0, I_1, u)\}.$$

Inserting equation 5.2 and 5.3, yields

$$\min_u \Bigg\{ \int_\Omega \int_\Omega w(x,y)(|u_1(x) - u_1(y)|_\varepsilon + |u_2(x) - u_2(y)|_\varepsilon) dy dx +$$

$$\lambda \int_\Omega \delta(x, u_0(x)) + (\nabla \delta(x, u_0(x)))^T (u(x) - u_0(x)) + (u(x) - u_0(x))^T (\nabla^2 \delta(x, u_0(x)))(u(x) - u_0(x)) dx \Bigg\}.$$

How to solve this equation is beyond the scope of this report, but the output of u generated by the libflow library will be used as the motion map M_x and M_y.

5.2.3 Invariance to camera movements

In chapter 3.4, it was declared that the system should be invariant to a limited amount of camera motion. The motions the system must be invariant to are camera pitch and yaw. The motion maps used for classification is the part of the system that is affected by this requirement the most. Two independent approaches has been proposed in order to eliminate the information introduced by camera rotation. The first approach, explained in the following, introduce a motion field shift with respect to the average motion between two frames, which eliminates most of the motion introduced by pitch and yaw rotations. The second approach suppress the motion introduced by the camera, by running HOG directly in the motion maps M_x and M_y[37].

Consider the motion vector map M from where M_x and M_y are derived. Each vector in M represent the estimated motion in which the pixel have moved from the subsequent frame to the current frame. This motion is the relative motion between the camera and the scene, which indicates the motion vectors as a combination of the camera motion and the objects in the scene. Let the motion vector at the point (x,y) in the motion vector map M be denoted m and the same point in the motion map M_x and M_y be denoted m_x and m_y. Figure 5.7(a) illustrates how the motion vector m can be seen as a vector addition of the camera motion m_c and the underlying scene motion m_s.

It is easy to see, that the camera motion vector m_c and the motion vector m need to be subtracted in order to extract the scene motion vector m_s. Since the camera motion vector m_c is unknown, one has to estimate it. Let p_b and p_o be the number of pixels in the frame belonging to the background and moving objects respectively. Under the assumption that there will be many more

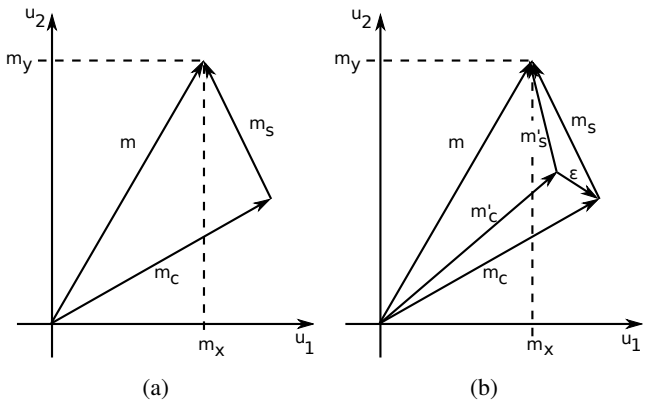


Figure 5.7: *Motion vector m can be split into a camera motion vector m_c and scene motion m_s, as in figure (a). In figure (b), the estimated scene motion m'_s is found by subtracting the estimated camera motion m'_c from the measured motion m. One should expect an error ε between the estimated scene motion m'_c and m_c.*

pixels belonging to a static background than to the moving objects, i.e. $p_b \gg p_o$. The average of the motion vector map M will be weighted almost exclusively by the motions generated by the pixels p_b and very little by the motions generated by the pixels p_o. Since the pixels p_b from the background move almost uniformly, they have a direct relation to the motion vectors m_c, hence one can estimate the approximate camera motion, m'_c, from the average of the motion vector map M.

$$m'_c = \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m(x,y) \tag{5.4}$$

where $\Omega_b \subseteq \Omega$ and $\Omega_o \subseteq \Omega$ are the subregions in the motion map domain $\Omega \subset \mathbb{R}^2$ containing the background pixels and the object pixel respectively. To find the scene motion m_s, one have to subtract the measured motion vector m with the estimated camera motion m'_c. Since the estimated camera motion is expected not to be equal the true camera motion, the error ε is introduced when using the estimated camera motion m'_c. m_s can be found as

$$m_s = m - m_c \tag{5.5}$$

$$m_c = m'_c + \varepsilon \tag{5.6}$$

$$m_s = m - m'_c - \varepsilon \tag{5.7}$$

but since ε is unknown, one can only estimate the scene motion m'_s as the scene motion m_s minus the error ε

$$m_s = m'_s - \varepsilon \tag{5.8}$$

By substituting the left hand side of eq. 5.8 with the expression from eq. 5.7, the estimated scene motion can be expressed as:

$$m'_s = m - m'_c.$$

m'_s is a good approximation to m_s, since the average error can be expressed as

$$\bar{\varepsilon} = \bar{m}_c - \bar{m}'_c \tag{5.9}$$

Where $\bar{m}_c$ is

$$\bar{m}_c = \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m_c(x,y) \tag{5.10}$$

and $\bar{m}'_c$ from equation 5.4, can be rewritten as

$$\bar{m}'_c = \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} (m_c(x,y) + m_s(x,y))$$
$$\bar{m}'_c = \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m_c(x,y) + \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m_s(x,y)$$

Since $m_s(x,y) = 0 : (x,y) \subset \Omega_b$, this yields

$$\bar{m}'_c = \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m_c(x,y) + \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\cancel{\Omega_b} \cup \Omega_o)} m_s(x,y) \tag{5.11}$$

Inserting equation 5.10 and 5.11 into 5.9, yields:

$$\bar{\varepsilon} = \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m_c(x,y) - \frac{1}{p_b + p_o} \sum_{(x,y)\subset(\Omega_b \cup \Omega_o)} m_c(x,y) - \frac{1}{p_b + p_o} \sum_{(x,y)\subset\Omega_o} m_s(x,y)$$
$$\bar{\varepsilon} = -\frac{1}{p_b + p_o} \sum_{(x,y)\subset\Omega_o} m_s(x,y)$$

Which is equivalent to

$$\bar{\varepsilon} = -\frac{p_o}{p_b + p_o} \frac{1}{p_o} \sum_{(x,y)\subset\Omega_o} m_s(x,y)$$
$$\bar{\varepsilon} = -\frac{p_o}{p_b + p_o} \bar{m}_s(x,y)$$

When the assumption $p_b \gg p_o$ holds then $\bar{\varepsilon} \approx 0$. Since the objects in this projects motion maps are animals which are not spatially dominant in the scene, the assumption $\bar{\varepsilon} \approx 0$ seems a valid approximation.

Let the estimated object motion vector map M'_s and the x- and y-component maps $M'_{s,x}$ and $M'_{s,y}$ be defined as,

$$M'_s = M - M'_c = M - \bar{M}$$
$$M'_{s,x} = M - M'_{c,x} = M_x - \bar{M}_x$$
$$M'_{s,y} = M - M'_{c,y} = M_y - \bar{M}_y$$

A scaled and offset version of the x- and y- component maps $M'_{s,x}$ and $M'_{s,y}$ will be used both in

the training and classification step, see chapter 5. How $M'_{s,x}$ and $M'_{s,y}$ are scaled and offset will be clarified in the following section.

5.2.4 Scaling of the motion vector maps

In order to effectively utilize the estimated object motion maps $M'_{s,x}$ and $M'_{s,y}$ as training data, they need to be cropped and saved as images, see chapter 3.2. Since images stored on a computer is stored in a compressed RGB 24-bit (8-bit per channel) format, in which the color value interval pr. channel is $0-255$, the motion maps $M'_{s,x}$ and $M'_{s,y}$ need to be quantized and clamped into the interval $0-255$ in order to preserve the motion information.

For convenience this color interval will in the following be referred as 3 floating point channels with the value interval 0.0 to 1.0, but before saving the data to the disk they are converted to the RGB 24-bit format.

As were explained in section 5.2.3, the motion map M_x and M_y were subtracted with their average in order to extract the estimated object motion maps $M'_{s,x}$ and $M'_{s,y}$. This implies that the motion maps $M'_{s,x}$ and $M'_{s,y}$ has zero motion at the value 0, and movement both on the positive and negative side, depending on movement direction. If those values are to be quantized and clamped to lie within the interval $0.0-1.0$, one have to divide the motion maps with an appropriate value d and add an offset 0.5 to move the average (no motion) up to the middle of the interval $0.0-1.0$.

$$M^*_{s,x} = 0.5 + \frac{M'_{s,x}}{d}$$
$$M^*_{s,y} = 0.5 + \frac{M'_{s,y}}{d}$$

An illustration of the scaling and offset of the motion vector m'_s is shown in figure 5.8. Every

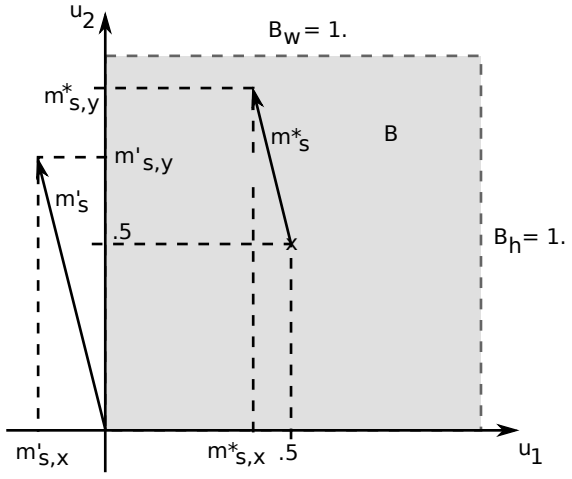


Figure 5.8: *The estimated object motion vector m'_s is scaled and offset with* 0.5 *into a vector m^*_s enclosed by the box B with width $B_w = 1.0$, height $B_h = 1.0$ and center in $(0.5, 0.5)$. The scaled object motion vectors' x- and y-components $m^*_{s,x}$ and $m^*_{s,y}$, will compose the scaled motion maps $M^*_{s,x}$ and $M^*_{s,y}$*

vector m_s^* that fit within the box B will be saved, where as any vector reaching outside the box, after scaling and translation, will be clamped to the box edge. The vector parts, reaching outside the box B will be seen as either errors due to noise in the estimated object motion map M_s' or motion so large and uncommon and the exact direction and length can be assumed less important to the classification process.

Since the scaling factor d is directly equivalent to the maximum pixel-velocity (pixels / frame) of the motion map that is measurable. The scaling factor d is determined by measuring the standard deviation of mean-corrected motion in a video sequence with legal camera motion, as defined in chapter 3.4. The scaling factor d is then set to 4 times the standard deviation. This should ensure the important object motions to be quantized within the interval $0.0 - 1.0$ where movements will lie in the interval

$$\begin{aligned} \text{left: } & 0.0 \leq m_{s,x}^* < 0.5 \\ \text{right: } & 0.5 < m_{s,x}^* \leq 1.0 \\ \text{down: } & 0.0 \leq m_{s,y}^* < 0.5 \\ \text{up: } & 0.5 < m_{s,y}^* \leq 1.0 \\ \text{stagnant: } & m_{s,x}^* = m_{s,y}^* = 0.5 \end{aligned}$$

Examples of the motion maps $M_{s,x}^*$ and $M_{s,y}^*$ are shown in figure 5.9(c) and 5.9(d) generated from the frames 5.9(a) and 5.9(b).

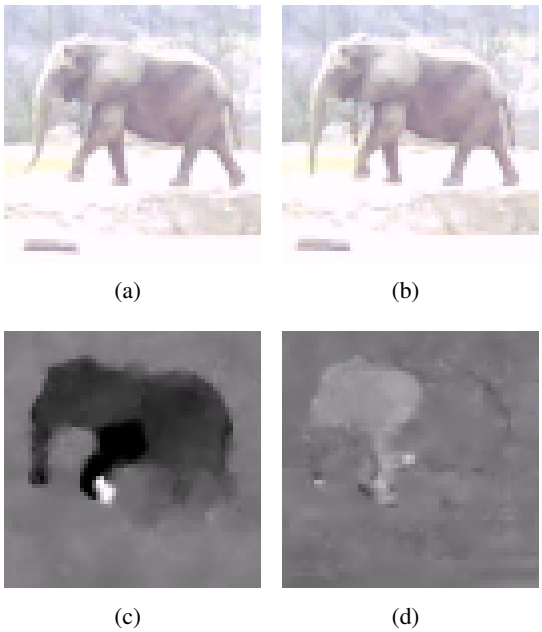


Figure 5.9: *Example of the object motion maps $M_{s,x}^*$ (c) and $M_{s,y}^*$ (d) generated from the two sequential frames (a) and (b). It is easy to see from (c), that the elephant must be walking left due to the dark gray color in $M_{s,x}^*$ and the head and ear is moving slightly upwards due to the slightly lighter color in $M_{s,y}^*$.*

As it was found in chapter 4.1.3, it is reasonable to assume that the detection performance of the 'action recognition' stage can be increased by using more than one motion vector map in combination. A way of utilizing the motion information from more than one motion vector map, the following section introduce a method which combines several motion vector maps into only one map, to be used with the HOG descriptor. These combined motion maps will from now on be called multi-motion maps.

5.2.5 Multi-motion maps

Note that in this subsection, the motion vector map M_s^* and its derivatives $M_{s,x}^*$ and $M_{s,y}^*$ will be referred to as M, M_x and M_y for simplicity.

As mentioned in chapter 4.1.3 using only one motion vector map M is most likely not enough to precisely encode an action of an animal. This is due to the fact that M from e.g. an elephant walking could for many frame-to-frame observations be very similar to an elephant standing. A case where a walking elephant is classified as a standing elephant is illustrated in figure 5.10. A

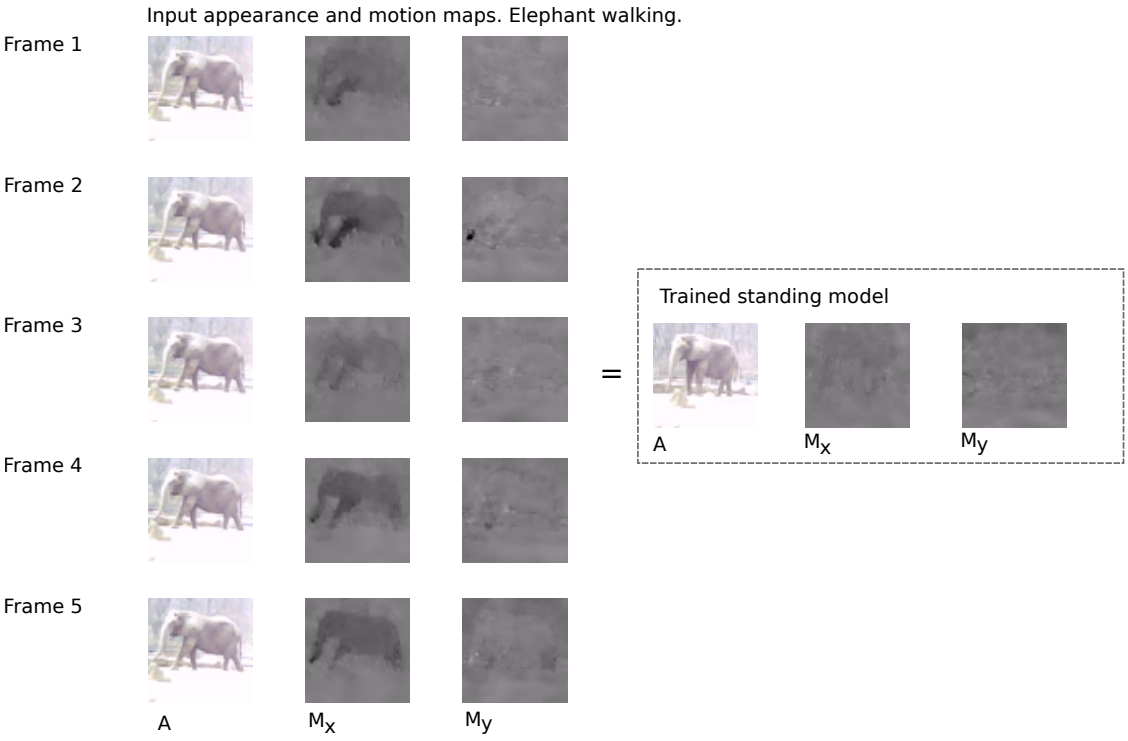


Figure 5.10: *Frame 3 of a sequence with a walking elephant is classified as a standing elephant, due to similar motion maps in both the M_x and M_y motion map.*

second problem with the use of only one motion map M, is the information loss of the motion behavior over a complete or partial motion cycle, e.g. a walking cycle. This information is crucial to distinguish e.g. walking from turning around.

To solve these issues an entity called the multi-motion map is proposed. Let a multi-motion map $\bar{M}$ be defined as a motion vector map with the same size as the motion vector map M, consisting

of an average of several motion vector maps. An illustration of the averaging process can be seen in figure 5.11. Using the average of several motion maps, lets $\bar{M}$ encode the movement within a given time interval connecting a sequence of motions to the action. See figure 5.12 for examples

Figure 5.11: *Example of M_x motion map being averaged over N frames in a sequence. The same approach applies to the M_y motion map.*

of averaged motion maps for a given video sequence with different values of N. As seen in figure

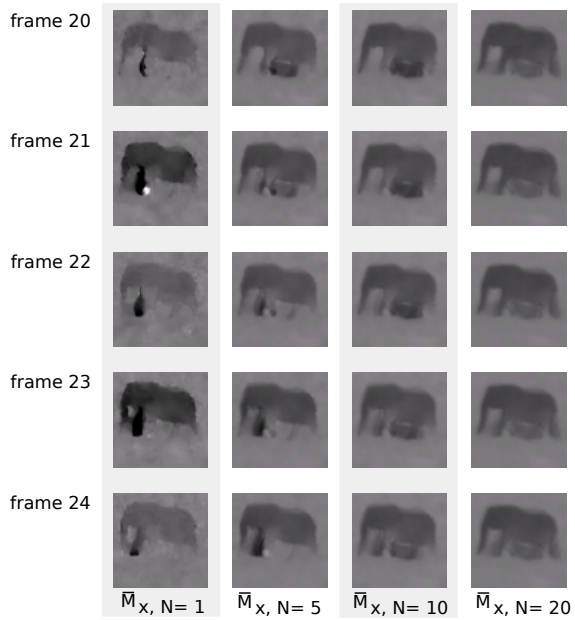


Figure 5.12: *The multi-motion map $\bar{M}_x$ for $N = 1, 5, 10, 20$. The sequence starts from frame 20, as the preceding frames were used to build the averaged motion map.*

5.12, averaging the motion maps have greatly reduced the noise in especially the background. Secondly it can be seen, that when N increases, the the multi-motion maps from different frames increase in similarity. This should greatly improve the action detection performance as cyclic and partial motions are encoded in the feature vector. As can be seen with the motion maps $\bar{M}_{x,N=10}$ and $\bar{M}_{x,N=20}$, the motion map seems to have covered most of the walking cycle for an elephant, i.e. the multi-motion maps are hardly changing from frame to frame.

It might be tempting just to choose N equal 20 for all the classes of actions in order to encode these cycles, but note that a running average over N samples works like a low pass filter, i.e. the greater N the more suppression is placed on the higher frequency temporal signals. Thus the drawback of choosing a too high value of N, is the loss of the faster or non-cyclic actions of an animal.

Due to the hard task of choosing an optimal N for all kinds of actions, it has been decided to test the classifier with various average motion maps generated with *N* equal 5, 10 and 20, see chapter 8.

5.2.6 Running average

Generating multi motion maps for training purposes is a straight forward task, but when applied to unknown data it is necessary to find a way to crop the needed motion map regions at runtime. To generate valid multi-motion maps, it is important that each multi-motion map is only updated with exemplars from the corresponding animal. For a low number N, registering invalid motion exemplars may severely corrupt the constructed multi-motion map and hinder correct action detection.
The cascaded classifier structure has been implemented so that it already pass detection windows from the first cascade to the second. To implement the multi-motion maps, we construct a search area for each detection window. The search area is a copy of the detection window to which we add a multi-motion map container, containing a running average of motion over the predefined number of frames. Each search area is extended by 10% of the detection window size and is stored for the next frame, where the new detection windows will be matched with the existing search areas. If a detection window is matched with a search area, the multi-motion map for the search area is updated with the motion map of the detection window and the new multi-motion map is used for action recognition. If a search region is not matched with a new detection window within a predefined number of frames, the search area is discarded, as it is considered obsolete. Correct action detection is not needed to succesfully build multi-motion maps, but require accurate detection windows from the previous cascade stage.

5.3 Histogram of oriented gradients

In order to encode the appearance and motion map features into a descriptor, it has been chosen to use histogram of oriented gradients (HOG) 4.1.2. HOG is a method of encoding features into descriptors such that e.g. supervised learning methods like the support vector machine (SVM) can be used to analyze and classify the descriptors [12]. The HOG descriptor implementation used is a library called 'INRIA Object Detection and Localization Toolkit (ODLToolkit)', which is an open source implementation based on the algorithm proposed by Dalal and Triggs [12][42].

The proposed classifier, see chapter 5, uses HOG descriptors for either the appearance images or the motion maps or both. Since the appearance and motion maps both are pixel-based images, they are easily used by the ODLToolkit without any changes to the library itself. But since the ODLToolkit does not support multiple types of images when calculating the descriptors, a lot of work has been put into improving the library to support input images of different types, allowing the appearance and motion-map images to be combined in one descriptor vector. Figure 5.13 illustrates the proposed extended work flow in addition to the original HOG work flow implemented in the ODLToolkit.

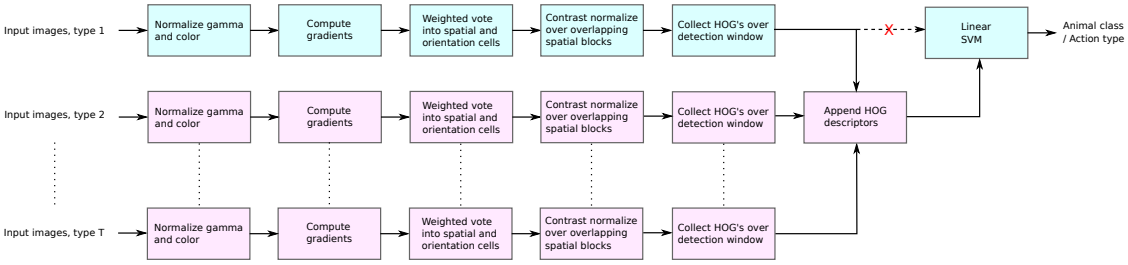


Figure 5.13: *The work flow of the HOG algorithm, where the teal colored boxes illustrates the original HOG algorithm. The magenta colored boxes illustrates the work done to extend the algorithm to combine HOG descriptors from T types of images, where the maximum allowable T depends on the amount of memory available on the system.*

As can be seen on figure 5.13, the HOG work flow has been extended to support T different types of input images. The support of multiple types of input images will be utilized to create classifiers based on both appearance and motion map images. Note that even though the motion map often is mentioned as only one type of input image, in the practical implementation the motion map type is composed of the x- and y-component of the motion vector map, and hence, the support of multiple input types is a must.

In the following, each step in the HOG algorithm will be explained. Keep in mind that the default parameters used for a standard person detector in the work done by Dalal and Triggs will be used throughout this work, if not mentioned otherwise. The reason why the default parameters are used is due to the big diversity in the appearance and composition of animals, which would make the search of the optimal parameters ambiguous.

5.3.1 Gamma/color normalization

In the original HOG algorithm proposed by Dalal and Triggs, they ended up using no image normalization at all. Their results showed only modest results on the detection performance. The conclusion was that the normalization step was not essential probably due to the block normalization in the later steps and due to the fact that they use gradients which are robust against brightness offsets.

5.3.2 Gradient computation

An objects appearance and shape has found to be characterized well by the distribution of local intensity gradients and / or edge directions. To find these local distributions, one needs to calculate the gradients for each pixel of the image. According to Dalal and Triggs, the best kernel yielding the best performing gradient, was the simple kernel [-1, 0, 1].

The gradient map will end up being computed in each pixel in the input image. Let $\Omega \subset \mathbb{R}^2$ be the image domain and $I(x,y)$ be the pixel value in the image coordinates $(x,y) \subset \Omega$. Let the kernel $K_x = [-1,0,1]$ and $K_y = [-1,0,1]^T$ be the kernels with which the image is convolved. Then the

computed gradient vector map G is found as

$$G = \begin{bmatrix} I * K_x & I * K_y \end{bmatrix}$$

From where the magnitude of the brightness change is found as

$$\|G\| = \sqrt{(G_1^2 + G_2^2)}.$$

5.3.3 Orientation Binning

In order to create the descriptors, the gradient vector map G is divided into blocks B of $2x2$ cells C of the size $8x8$ vectors, see figure 5.14. A local histogram H is made from each cell of gradi-

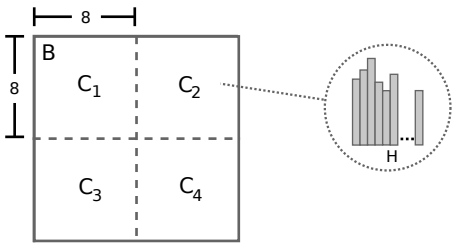


Figure 5.14: *The gradient vector map is divided into blocks B of* $2x2$ *cells C of the size* $8x8$*. For each cell C, a historgram is generated with* 18 *bins covering the interval* $0° - 180°$.

ents in terms of the gradient orientation and magnitude. The orientations are truncated into bins and the magnitude is accumulated to the bin in question. Dalal and Triggs use histograms of 9 bins covering orientations in the interval $0° - 180°$, where each bin covers $20°$. The gradients are binned into the $0° - 180°$ span, by ignoring the sign of the gradient. To minimize problems of orientations close to bin borders, a bilinear interpolation method is applied to the bin voting system.

Since this project is aiming at using motion maps, see chapter 5, it is important to be able to distinguish if an animal is walking left or right, which is directly encoded in the motion maps5.2.3, where light pixels indicates an animal is walking to the right and dark pixel the opposite. Since the sign of the gradient of the motion maps shows this exact information, it has been decided to truncate the orientations of the gradients into 18 bins covering the interval $0° - 360°$.

5.3.4 Normalization of the descriptor blocks

Since the gradient magnitudes vary over a great range due to local variations and contrast between background and foreground, Dalal and Triggs found it necessary to do normalization in order to increase the detection performance. The proposed normalization is to normalize each cell histogram H against all the other histograms in a block B. Let v be the unnormalized descriptor vector and $\|v\|_2$ be the L2-norm of the vector v. Then the normalization is found as

$$v = \frac{v}{\sqrt{\|v\|_2^2 + \varepsilon^2}}.$$

In order to increase the detection performance even further, the blocks are overlapped. In this way each cell contribute to the descriptor vector multiple times, but being normalized with respect to different blocks. Dalal and Triggs found that a block stride of 8 yielded the best detection performance. Figure 5.15 illustrates the four overlapping blocks, where the mid cell is contributing to the resulting descriptor vector four times.

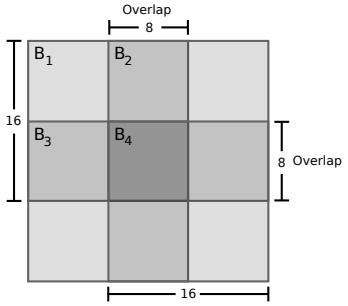


Figure 5.15: *The blocks are arranged with a stride of 8. This yields an overlap of the blocks, causing the mid cell to be covered by four different blocks.*

It has been shown that weighing down the edge pixels, using a Gaussian spatial window over each pixel in a block, before accumulating orientation votes into cells improve performance.

5.3.5 Collect and append HOG descriptors over the detection window

To show how the HOG descriptors are appended over an image or detection window, it has been decided to give an example how to calculate the descriptor size from an input example. The size of the descriptor vector depends on the block size, the histogram bin size, the cell size, the block stride and the input image size. A practical example of the correlation between these parameters and different types of input images, is given in figure 5.16. The example illustrates three types of input images of the size 64*x*64 consisting of an appearance frames, x-motion and y-motion maps. The block grid is arranged with a cell size 8*x*8, histogram bin size 18, block size 2*x*2, block stride 8 in which the HOG descriptors are calculated in each input image. The size of the resulting descriptor vector can be calculated as follows.

Let the image width and height be I_w and I_h, block width and height B_w and B_h and the block stride B_s, then the total amount of blocks in the input image can be found with the formula

$$\text{Number of blocks} = \left(\frac{I_w - B_w}{B_s} + 1\right)\left(\frac{I_h - B_h}{B_s} + 1\right) = \left(\frac{64-16}{8} + 1\right)\left(\frac{64-16}{8} + 1\right) = 49$$

If each cell contributes with a descriptor size equal to the bin size 18 and each block contains four cells, the descriptor size can be found as

$$\text{Descriptor size} = 49 \cdot 4 \cdot 18 = 3528$$

Since there is three types of input images in this example, the resulting appended descriptor size is $3 \cdot 3528 = 10584$.

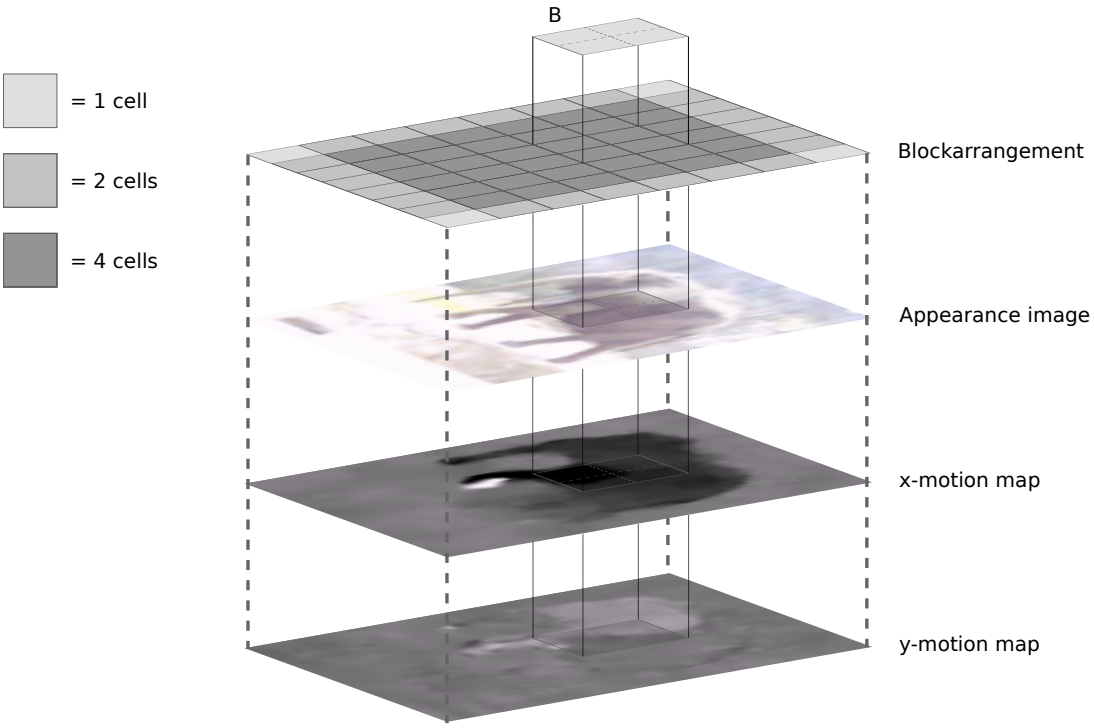


Figure 5.16: *Example of how the blocks are arranged with respect to the input images. The block overlaps are illustrated with different values of gray, where dark gray equals an overlap of four cells, gray equals an overlap of two cells and light gray equals no overlap. The block B is separated into regions illustrating the cells.*

5.4 Support Vector Machine

In chapter 5.1 it was found that Support Vector Machine classifiers (SVM) [43] should be used in a cascade. Since the goal of this project is not to develop a SVM classifier, it has been chosen to use the free and open source SVM library lightSVM [44]. The library comes with the commandline tool `svm_learn`, which has been used to train classifiers from the training data generated with the HOG decriptor clarified in chapter 5.3. An illustration of the training of the classifiers is shown on the figure 5.17.

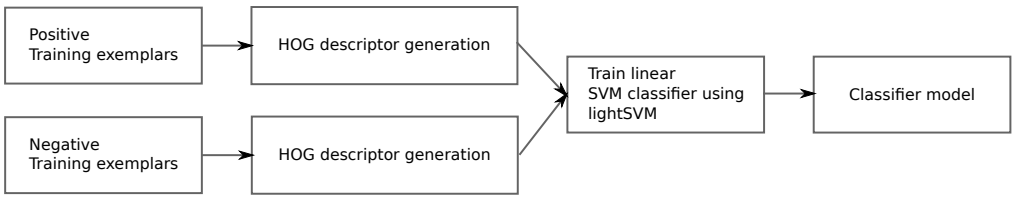


Figure 5.17: *The positive and negative exemplars are used to generated HOG descriptors, which is given to the lightSVM library. This produces a classifier model, which is used in the classifiers used by the main application.*

The resulting Classifier models are used in the classifiers used by the main application. It is has been chosen to use linear SVM classifiers to separate positive from negative exemplars due to the good detection performance achieved by the human detection project by Dalal and Triggs[12],

which is also using HOG descriptors.

Since the SVM classifier is a very important element in this project, it has been chosen to shed some light on the actual SVM algorithm and its capabilities.

5.4.1 Derivation of the SVM classifier

The SVM classifier is a supervised classifier, which entail the need of some prelabeled training data. The training data in this project would e.g. consists of

1. Training labels $y \subseteq \{-1,+1\}$, where $y = +1$ means e.g. an elephant and $y = -1$ means e.g. not an elephant

2. Training data $x \subseteq \mathbb{R}^d$, where the components of x is notated as follows $x = \{x^1,...,x^d\}$. The training data would in this project be the HOG descriptors extracted from training exemplars of appearance and motion frames.

Given these training data, the idea is to find a classification function $F : \mathbb{R}^d \rightarrow \{-1,+1\}$ which is able to classify any new data x. Such a function can be expressed as

$$F(x) = sign\, f(x)$$

with a decision function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ parameterized as

$$f(x) = a^0 + a^1x^1 + a^2x^2 + ... + a^dx^d.$$

If the coefficients $a^0,...,a^n$ are written as $w = (a^0,...,a^n) \subseteq (R)^{d+1}$ and augmenting x with the constant 1 in the first position $\tilde{x} = (1,x^0,...,x^d) \subseteq \mathbb{R}^{d+1}$ $f(x)$ can be written as

$$f(x) = w^T\tilde{x}. \tag{5.12}$$

From now on, $\tilde{x}$ will be referred as x in order to simply things.

$f(x)$ can be visualized as the hyperplane $w^T\tilde{x} = 0$ separating the negatives from positives, see figure 5.18.

The idea is to find the parameters w, which separates the training data the most, i.e. the parameters w, which maximizes the distance p from the hyperplane to the nearest data on each side. The distance p can be found as

$$p = \min_{i=1,...,n} \left| \left(\frac{w}{||w||} \right)^T x_i \right|,$$

for all the training samples n.

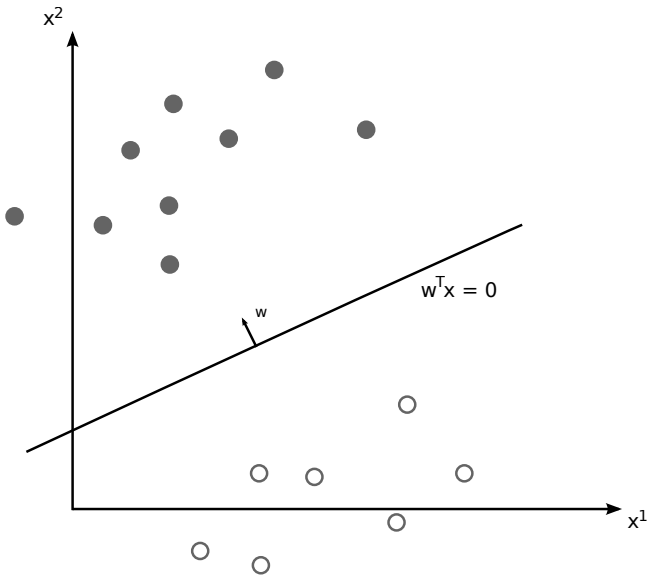


Figure 5.18: *The hyperplane separating the two classes.*

Given the above definitions, the parameters w can be found with the following optimization problem

$$\max_{w \subseteq \mathbb{R}^d} p$$

subject to

$$p = \min_{i=1,\ldots,n} \left| \left(\frac{w}{||w||} \right)^T x_i \right|$$

and

$$\operatorname{sign} w^T x_i = y_i, \quad \text{for} \quad i = 1, \ldots, n.$$

The minimization of p can be rewritten to

$$p \leq \left| \left(\frac{w}{||w||} \right)^T x_i \right|,$$

which constraints p to be smaller than all $\left| \left(\frac{w}{||w||} \right)^T x_i \right|$. This forces p to be a positive number.

If the equality constraint is inserted in the inequality constraint and divided by p, the optimization problems can be written as

$$\max_{w \subseteq \mathbb{R}^d, p \subseteq \mathbb{R}^+} p \tag{5.13}$$

subject to

$$y_i \left| \left(\frac{w}{p||w||} \right)^T x_i \right| \geq 1, \quad \text{for} \quad i = 1, ..., n,$$

where R^+ stands for all positive values.

Using that $||w|| = \frac{1}{||p||}$, the maximization 5.13 can be written as the minimization of $||w||$ or the equivalent solution $||w||^2$, which is easier to solve numerically. The optimization problem can at last be written as

$$\min_{w \subseteq \mathbb{R}^d} ||w||^2 \tag{5.14}$$

subject to

$$y_i \left| w^T x_i \right| \geq 1, \quad \text{for} \quad i = 1, ..., n,$$

The above approach is also called the maximum-margin classifier. An illustration of positives and negatives separated by the two margin hyperplanes is found in figure 5.18. Until now, the

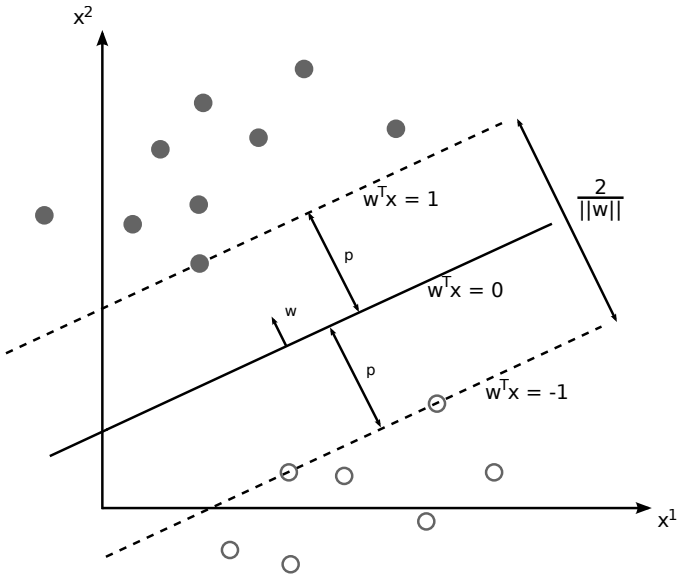


Figure 5.19: *The hyperplane and its margins separating the two classes.*

maximum-margin classifier has been found as an optimization problem 5.14. The solution to this problem does only exist if the data is completely separable. Usually data will not be completely separable due to outliers, see figure 5.20.

To overcome this problem, the soft margin classifier is introduced.

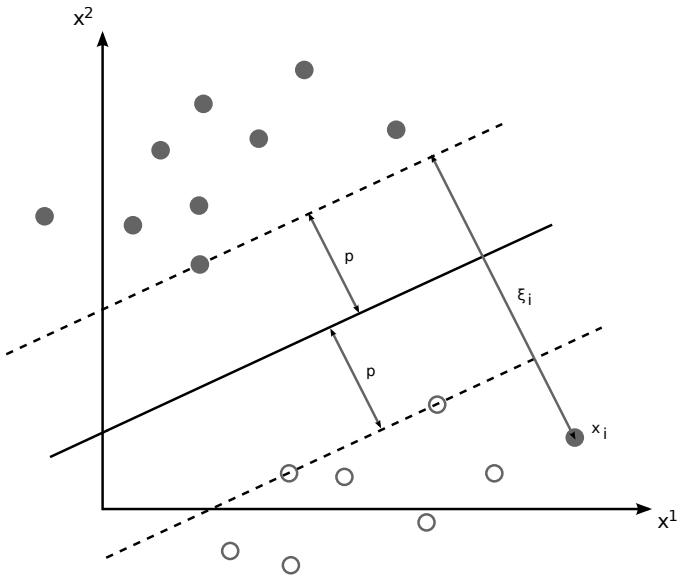


Figure 5.20: *An outlier violates the margins by ξ_i.*

Soft margin classifier

In order to find the parameters w even though there exists outliers, the slack variable ξ_i is introduced yielding the a new optimization problem defined as

$$\min_{\substack{w \subseteq \mathbb{R}^d \\ \xi_1,...,\xi_n \subseteq \mathbb{R}^+}} \quad ||w||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{5.15}$$

subject to

$$y_i \left| w^T x_i \right| \geq 1 - \xi_i, \quad \text{for} \quad i = 1,...,n,$$

where $C > 0$ is a regularization variable, which can be used to balance between robustness and correctness. Increasing C will increase the impact of the slack variable ξ_i and making the solution similar to the original max margin optimization problem in 5.15. Decreasing C will make ξ_i having a decreasing impact on the solution and greater outliers are accepted. The effect of the regularization variable C is illustrated in figure 5.21.

Non-linear data classification

In some situations, the data which needs separation is not linear and hence, the maximum margin classifier can not separate the data. To cope with that situation one can transform the non-linear data into a linear space before solving the optimization 5.14. This transformation or mapping (possible non-linear) is defined as $\varphi : \mathbb{R}^d \to \mathbb{R}^m$, which change the decision function 5.12 to

$$f(x) = w^T \varphi(x), \tag{5.16}$$

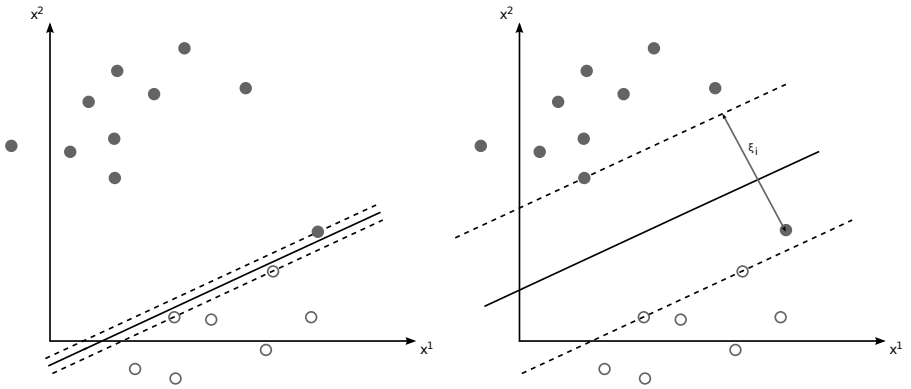


Figure 5.21: *Example where robustness might be more attractive than correctness. The regularization variable C in (a) is greater than the one in (b). It is easy to see that (b) illustrates a more robust classifier than the more correct classifier illustrated in (a).*

where $w \subseteq \mathbb{R}^m$ is the solution to the optimization

$$\min_{\substack{w \subseteq \mathbb{R}^m \\ \xi_1,...,\xi_n \subseteq \mathbb{R}^+}} \quad ||w||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{5.17}$$

subject to

$$y_i \left| w^T \varphi(x_i) \right| \geq 1 - \xi_i, \quad \text{for} \quad i = 1,...,n.$$

An example of a useful mapping from the cartesian coordinate system to the polar coordinate system is defined as

$$\varphi : \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \frac{\sqrt{x^2+y^2}}{arctan\frac{y}{x}}.$$

The illustration on figure 5.22 shows the effect of mapping the data set into a polar coordinate system.

It is clear that the mapping to the polar coordinate system enables the maximum margin classifier to separate the data set.

Since there is no restrictions on the mapping $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^m$, m can be of much higher dimensions compared to d and might introduce some computational and runtime issues due to the increased amount of used memory.

The above concerns are basically unfounded due to kernelization, which will be shown in the following.

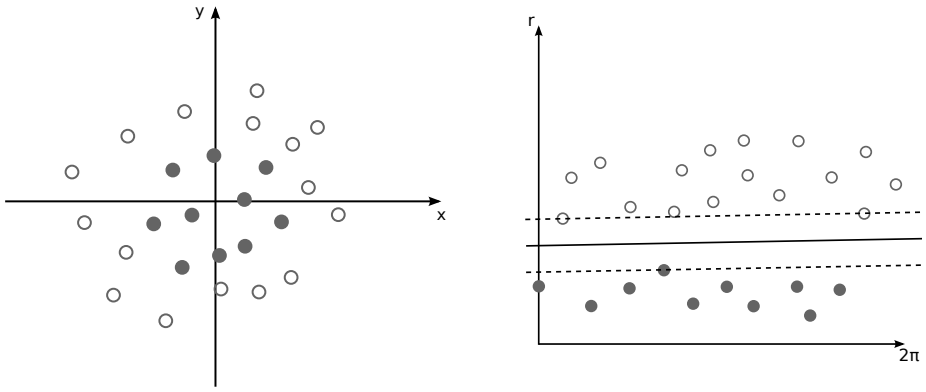


Figure 5.22: *Example of a mapping from the non-linear data set in the cartesian coordinate system (a) into the polar coordinate system (b). When the data set is mapped into the polar coordinate system, it is possible to separate the data with a maximum margin classifier.*

The representer theorem

In the following The representer theorem [43, p. 16] , it is shown that w can not be arbitrary, but it lies in a subspace of $\mathbb{R}^m$ with dimensions n or less, where n is the number of training samples. The proof of the representer theorem will not be examined in this chapter, but it basically says that w can be represented as follows

$$w = \sum_{j=1}^{n} \alpha_j \varphi(x_j) \quad \text{for coefficients } \alpha_1, ..., \alpha_n \subseteq \mathbb{R}. \tag{5.18}$$

Substituting w in the optimization problem 5.17 with the equation 5.18, yields the new optimization problem

$$\min_{\substack{\alpha_1, ..., \alpha_n \subseteq \mathbb{R} \\ \xi_1, ..., \xi_n \subseteq \mathbb{R}^+}} \sum_{i,j=1}^{n} \alpha_i \alpha_j \varphi(x_i)^T \varphi(x_j) + \frac{C}{n} \sum_{i=1}^{n} \xi_i \tag{5.19}$$

subject to

$$y_i \sum_{j=1}^{n} \alpha_j \varphi(x_i)^T \varphi(x_j) \geq 1 - \xi_i, \quad \text{for} \quad i = 1, ..., n,$$

and the generalized decision function becomes

$$f(x) = \sum_{j=1}^{n} \alpha_j \varphi(x_i)^T \varphi(x_j).$$

The general idea of the representer theorem, is that most of the coefficients α_j will be zero and hence, the computational and runtime costs are reduced. All the training examples x_j with non-zero coefficients α_j are usually called support vectors. It should also be noted that instead of optimizing over $w \subseteq \mathbb{R}^m$ one can optimize over $(\alpha_1, ..., \alpha_n) \subseteq \mathbb{R}^n$.

Even though the representer theorem has reduced the computation and runtime requirements, one still has to calculate and store $\{\varphi(x_1),...,\varphi(x_n)\}$ before finding the parameters of the maximum margin classifier. In order to avoid this bottleneck, kernelization comes into play.

Kernelization

By observing the optimization problem 5.19, it is easy to see that the transformed features are represented pairwise. This property can be utilized by defining the kernel parameter $k(x,x') = \varphi(x)^T\varphi(x')$. By estimating the kernel directly instead of the individual $\varphi(x_i)$ parameter, the number of needed calculations reduced to $\frac{n(n+1)}{2}$ unique values due to symmetry, instead of the $n \cdot m$ entries of the vectors $\varphi(x_i)$.

Inserting $k(x,x')$ into the optimization problem 5.19, yields the kernelized maximum margin classifier, also called Support Vector Machine. The SVM classifier is found by solving the following optimization problem

$$\min_{\substack{\alpha_1,...,\alpha_n \subseteq \mathbb{R} \\ \xi_1,...,\xi_n \subseteq \mathbb{R}^+}} \sum_{i,j=1}^{n} \alpha_i\alpha_j k(x_i,x_j) + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{5.20}$$

subject to

$$y_i \sum_{j=1}^{n} \alpha_j k(x_j,x_i) \geq 1-\xi_i, \quad \text{for} \quad i = 1,...,n,$$

The SVM approach by using the kernelization $k(x_i,x_j)$ instead of the individual $\varphi(x_i)$ approach have two major advantages.

1. In some cases, calculating an impression of $k(x_i,x_j)$ might give some speed improvement. An example could be the square of the mapping $\varphi : x \to (1,\sqrt{2}x,x^2) \subseteq \mathbb{R}^3$ versus the equivalent kernelized version $k(x,x') = (1+xx')^2$.
2. One can construct kernel functions $k(x_i,x_j)$ without knowing how to compute φ.

In general, $k(x,x')$ gives a lot of opportunities to design different kernels, as long as Mercer's condition is fulfilled, see [43, p. 19-20]

Mercer's condition:
Let X be a non-empty set. For any positive definite kernel function $k : X \times X \to \mathbb{R}$, there exists a Hilbert space H and a feature map $\varphi : X \to H$ such that

$$k(x,x') = (\varphi(x)^T\varphi(x'))_H,$$

where $(\cdot)_H$ denotes the Hilbert space.

Examples of some kernels commonly used by SVM's are shown in the following

- Linear: $K(x,x') = x^T x'$
- Polynomial: $K(x,x') = (1 + x^T x')^m$
- Gaussian: $K(x,x') = exp\left(-\frac{||x-x'||^2}{2\sigma^2}\right)$ with $\sigma > 0$

which all obey the Mercer's condition.

In this project the linear kernel will be used, which has been proven to perform well with HOG descriptors [12].

PART III

VERIFICATION

Chapter 6

Methodology

In this part of the report, the system performance is tested regarding animal detection & recognition and action recognition, within the system scope given in 3.4. Since the 'animal detection and recognition' is the first stage of the proposed cascade classifier, it is highly influential on the performance of the second stage of the cascade, the 'action recognition' stage. To avoid punishing the second stage for errors in the first stage, only the regions detected as the animal in question in the first stage, will be evaluated in the second stage.

The stages 'animal detection and recognition' and 'action recognition' will in the following chapters be verified and tested. In each chapter, the type of test which is about to be conducted, choice of descriptor combinations, the setup and the training and verification data will be clarified before the performance results of the specific stage is presented.

6.1 Verification data notes

To verify the 'action recognition' stage of the classifier, a dataset of different animals performing different actions is needed and must be recorded using a moving camera. Unfortunately such a dataset is not publicly available. We visited Pittsburgh zoo on several occasions to built such a dataset, but only very little useful data came from it. Our dataset for training and verifying the action recognition stage only contain elephants with the actions walking left or right. We verify our algorithm against this dataset to validate that our action detection algorithm is not significantly affected by a moving camera.
The dataset does not contain enough different actions to validate the action recognition algorithm itself. We have therefore chosen to train and validate our algorithm against the KTH dataset also, which provides six different action sets performed under four different conditions and includes data for 25 different persons.We use the same data segregation as Schuldt, Laptev and Caputo[45], using persons 11 to 18 for training and persons 1, 4, 19 to 25 for validation. Our elephant data set contains one sequential movie containing 10560 frames, frames 0-4999 are used for training and frames 5000 and up are used for validation.

Chapter 7

Animal detection and recognition

In order to test the 'animal detection and recognition' stage, the detected location of the animal will be evaluated with respect to the overlap of the verification data. Let the verification region be noted W and the detected region be noted D, then the overlap O can be found as the intersection between W and D over the union of W and D,

$$O = \frac{W \cap D}{W \cup D}.$$

The overlap measure gives an indication of how precise the system is at estimating the correct location of the animals in the image. The overlap of the detected location of an animal versus where you see the animal in the image is one of the key factors of a good user experience.

Another and maybe more important key factor of a good user experience, is the ability to detect the animal in the scene versus falsely detecting an animal not present in the scene. This measure is often called a Receiver Operation Characteristic curve (ROC curve) and found as the true positive rate, or recall, versus the false positive rate. Recall and false positive rate is calculated as follows:

$$recall = \frac{true\_positives}{true\_positives + false\_negatives}$$

$$false\_positive\_rate = \frac{false\_positives}{true\_negatives + false\_positives}$$

The ROC curve is generated by stepwise adjusting the threshold deciding whether the region is positive or not. With a low threshold, the classifier will classify more regions as positives, increasing the number of false positives and lowering the false negatives. The lower threshold will also result in more false positive regions, many clustered near the actual positives, resulting in many detections with low overlaps. On the other hand, increasing the threshold will decrease the detection performance, but at the same time reject more regions with a low overlap. That is, to find the optimal 'animal detection and recognition' stage one must find the best balance between a good detection rate over false positive rate versus keeping the overlap between the ground truth data and the detected regions as large as possible. This is done by evaluating a set

of different detected regions with different overlaps, in order to find the minimum acceptable overlap for a system like this. When the minimum acceptable overlap has been found, for all the positive regions, each positive region having an overlap less than the chosen minimum overlap, will be discarded and changed to a false positive. Changing a detected region from true positive to a false positive, will generate one additional false negative and one less true negative, as the verification region is changed from found to missed.

7.1 Choice of descriptors

The 'animal detection and recognition' stage of the classifiers main functionality is to locate and classify the type of animals being observed. As has been explained in the chapter about the HOG algorithm, chapter 5.3, the ODLToolkit has been extended to enable it to combine multiple input images into one descriptor. This enables the use of any combination of both the appearance and motion descriptors to detect and recognize the animal. The descriptor made from the appearance images are good for encoding texture, shape and color compositions, which seem to be many of the indicators humans use to recognize animals. The descriptors made from motion, only encode motion patterns and to some degree shape contours of the moving parts. Since many animals have very similar movements or may be completely still on many occasions, the use of motion descriptors seem less attractive than the appearance descriptors for the 'animal detection and recognition' stage.

The above reasoning leads to the test of a classifier based on only the appearance descriptors and a classifier based on a combined appearance and motion descriptor, to see how motion information affect performance.

7.2 Setup

The classifier based on appearance descriptors, will be using descriptors generated by the HOG algorithm using the following parameters:

- Cell size: $8x8px$.
- Block size: $4x4px$ cells.
- Linear gradient voting into 18 orientation bins in $0° - 360°$.
- Block spacing stride of 8 pixels.
- $64x64px$ detection window.
- Gaussian spatial window, with $\sigma = 8 = 0.5 \cdot block\ width$.

The cell size, block size, spatial Gaussian window size and the block spacing stride has been chosen on the basis of the experiments done by Dalal and Triggs[12]. The parameters were found to be optimal for pedestrian detection, but would most likely perform similarly on animals.

Since the intention of this system is to detect and classify different species of animals, it would not be feasible to find the optimal parameters for each type of animal. The same reasoning is behind the choice of a detection window size of $64x64px$, since a squared detection window is the best compromise for detecting both tall and wide animals. By using the same detection window for all animals, much computational performance can be gained. Empirical experiments show that the size of $64x64px$ is appropriate, as most animals of interest are $64x64px$ or larger in the images.
For linear gradient voting, 18 bins in $0^\circ - 360^\circ$ will be used as animals often have similar textures and colors, i.e. 'signed' gradients should help substantially detecting the animal from the background, as Dalal and Triggs observed was the case for e.g. cars and motorbikes.

The classifier based on appearance and motion descriptors will be using the same parameters as above, except the use of only 9 orientations bins in $0^\circ - 180^\circ$, i.e. the 'sign' of the gradient is ignored. Using 'unsigned' gradients may help detecting animals from their motion information, even though the motion map exemplars, generated from the animals motion, are direction sensitive. See the figure 7.1.

(a) (b)

Figure 7.1: *Motion map exemplars generated from the x-component of a motion map vector, from an elephant walking left (a) and right (b).*

Since the two training exemplars illustrated in the above figure represents the same class Elephant and both are used to train a classifier, it is clear that 'unsigned' gradients should help making the two exemplars more comparable.

7.3 Training and verification data

The training data used to train the 'animal detection and recognition' stage of the classifier, is as mentioned above both appearance and motion map exemplars. The appearance exemplars have been cropped from videos recorded with the HMD in Pittsburgh Zoo & PPG Aquarium [46]. The motion map exemplars have been cropped from two motion map videos generated from the appearance video, the x-component and y-component of the motion vector map respectively. All the cropped exemplars have been cropped and scaled to the same size as the detection window of $64x64px$. The training exemplars contains images of elephants from different poses, angles and actions. The video has been recorded under natural varying light conditions and with low to medium camera movements.

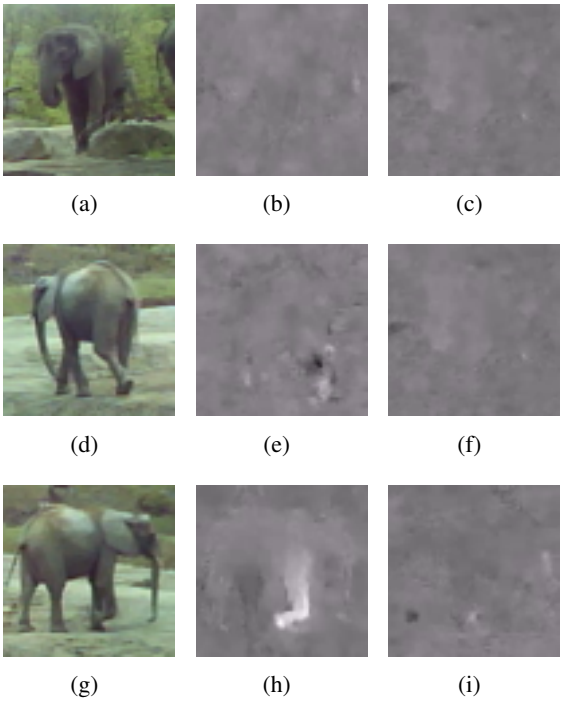


Figure 7.2: *Examples of the training data used to train the 'animal detection and recognition' stage of the classifier. In the case of the appearance only classifier, the exemplars (a), (d) and (g) will all be encoded as three individual HOG descriptors. In the case of the appearance and motion classifier, three HOG descriptors will be assembled from the exemplars [(a) (b)(c)], [(d) (e) (f)] and [(g) (h) (i)].*

Three examples of exemplars from the training data is shown in figure 7.2, from which three sets of descriptors will be generated.

Notice the lack of motion information in the exemplars 7.2(b), 7.2(c), 7.2(e) and 7.2(f) when the elephant is standing still. This could definitely cause some problems regarding distinction between the background and the elephant.

The training exemplars have been cropped from different sequences of consecutive frames within the frame interval 0 - 4999 in the recorded elephant movie. Selected sequences of frames containing elephants in the rest of the elephant movie, in the interval 5000 - 10560 frames, is used to label verification data from where 1530 regions have been labeled as containing an elephant. Negative exemplars have been randomly cropped from the video, which does not contain elephants. The classifier based on appearance HOG descriptors has been trained on the amount of exemplars shown in table 7.1 and the classifier based on both appearance and motion map exemplars has been trained on the amount shown in table 7.2. Each amount of positive training exemplars have been generated with corresponding left-right reflections. The negative exemplars have been randomly cropped from non-elephant frames, from the same video. After the first classification run, several false-positives has been collected as hard negative exemplars. The hard negatives have been used to train the final classifiers.

	Appearance
Positive elephant exemplars	4942
Negative exemplars	6113

Table 7.1: *Training exemplars used to produce HOG descriptors for the appearance classifier.*

	Appearance	Motion-x	Motion-y
Positive elephant exemplars	4942	4942	4942
Negative exemplars	6113	6113	6113
Hard negative exemplars	1564	1564	1564

Table 7.2: *Training exemplars used to produce HOG descriptors for the appearance and motion classifier.*

7.4 Results

The goal is to find the optimal balance between a high recall over false positive rate versus keeping the minimum overlap between the ground truth data and the detected regions as large as possible for the cascade stage.

In order to show what impact the minimum acceptable overlap has on the detection performance, figure 7.3 shows the ROCs from different minimum overlaps of the appearance classifier. The combined appearance and motion classifier will not be shown in this context due to bad results, which will be clarified later in this chapter.

To find the ideal minimum acceptable overlap and thereby the resulting ROC curve, the detection results were manually inspected. Figure 7.4 shows four examples and their respective overlap.

The choice of which detected region is acceptable and which is not, is completely subjective, and might differ from person to person. We suggest the following reasoning:

Figure 7.4(a) shows only the back of an elephant, most of the head outside the detected area, and therefor not found acceptable. Figure 7.4(b) illustrates the classifier detecting a small elephant partly occluding a bigger elephant. Due to the bigger elephant, the detected region is significantly larger than the ground-truth region for the smaller elephant and it is not clear which elephant is detected. Since the box does not clearly favor one or the other, the detection should be discarded. Figure 7.4(c) shows the classifier detecting an elephant missing a small part of the head. Since the most of the elephant is within the detected region, it is found acceptable. Figure 7.4(d) shows a nearly perfect detection, which obviously should be accepted.

On the basis of the detected regions, a minimum overlap of 0.6 is found suitable. Consequently for each detected region below an overlap of 0.6, the true positive label will be discarded and generate one extra false positive, one extra false negative and one less true negative, which will impact the ROC curve in a negative direction. Figure 7.5 illustrates an example of the 0.6 bound-

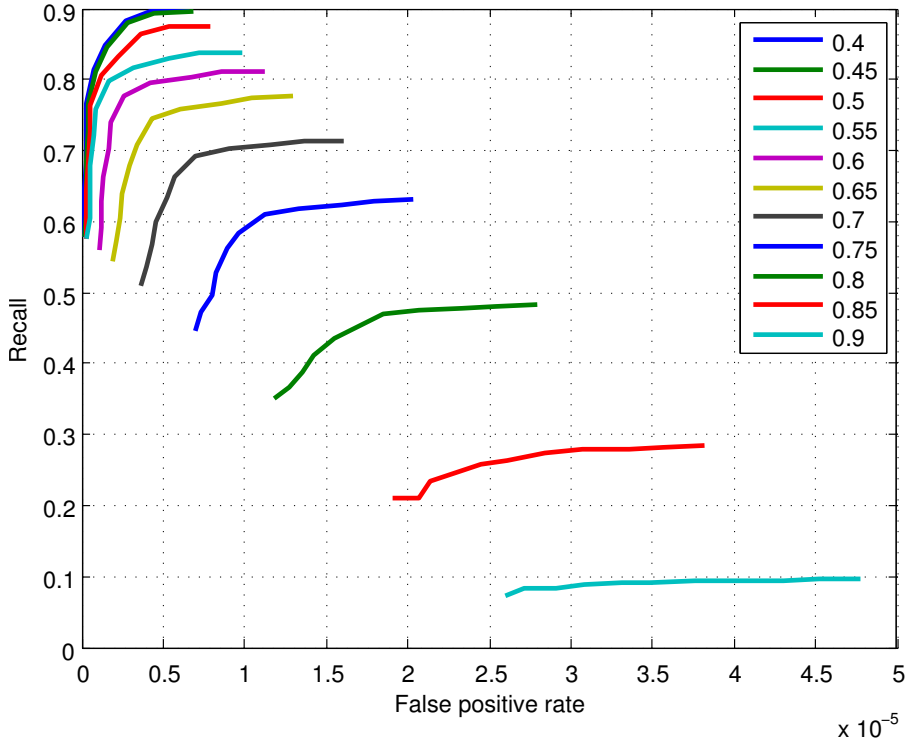
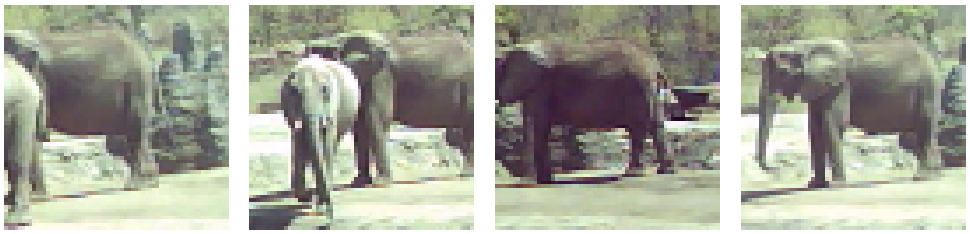


Figure 7.3: *Each ROC curve has been generated from testing* 1530 *regions in the verification frames, each with a different minimum acceptable overlap.*



(a) 0.5 overlap (b) 0.59 overlap (c) 0.67 overlap (d) 0.88 overlap

Figure 7.4: *Four different detections and their respective overlap with the verification labels.*

ary in comparison with the overlap of all the detected regions for a threshold of 0.4.

Using the minimum overlap of 0.6 and sweeping the threshold from 0.1 to 1.1, generates the ROC curve for the appearance only classifier on figure 7.6 and the ROC curve for the combined appearance and motion classifier on figure 7.7.

The ROC curve for the combined appearance and motion classifier has a significantly worse recall than the appearance only classifier. At a false positive rate of $1.1x10^{-4}$ the classifier only detects a little over a third of the ground-truth verification labels. This suggests that the motion data in combination with the appearance data is not good for detecting the animal. A reason for this may be the large variation in the motion data for e.g. an elephant standing versus an elephant walking. Appearance-wise there is not much difference of an elephant walking versus

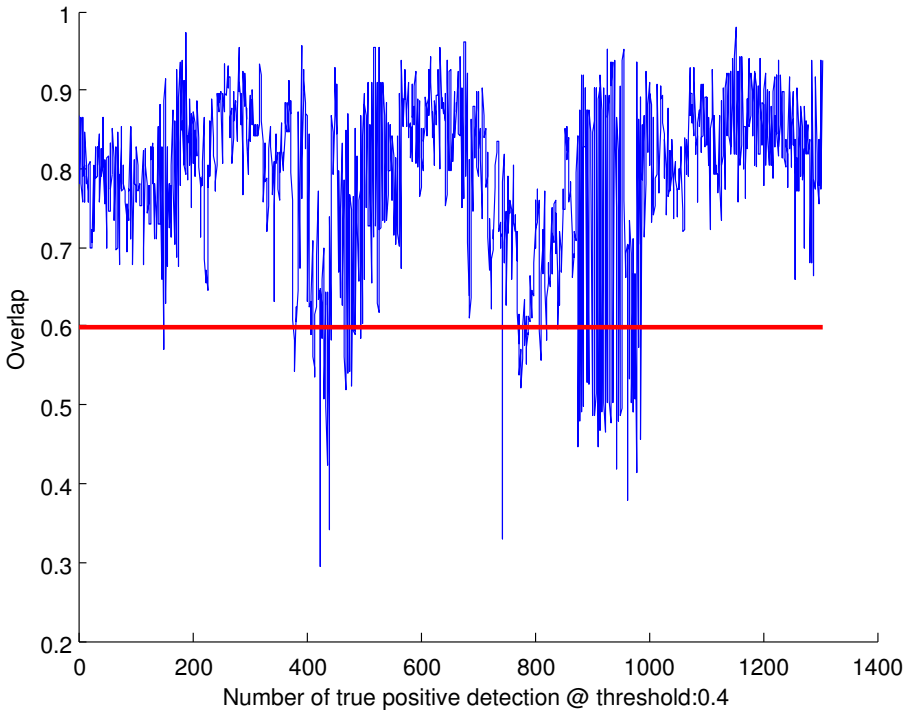


Figure 7.5: *The overlap between the detected region and the verification region for each true positive detection at a threshold of* 0.4*. The red line illustrated the minimum overlap of* 0.6*. Every true positive detections below that boundary will be discarded and generate one extra false positive, one extra false negative and one less true negative.*

an elephant standing, as the overall shape does not change significantly. This generates a lot of descriptors with very little in common for the SVM classifier to classify, and thus giving the bad results.

The ROC curve for the appearance only classifier looks more promising. This indicates that a classifier based only on the appearance desciptor is better at detecting an animal rather than a classifier based on both appearance and motion descriptors, which agrees with the intuition given in the introduction of this chapter. The threshold 0.4 yielding the recall of 0.7948 at a false positive rate of $4.249x10^{-6}$ will be used in the verification of the 'action recognition' stage in the next chapter.

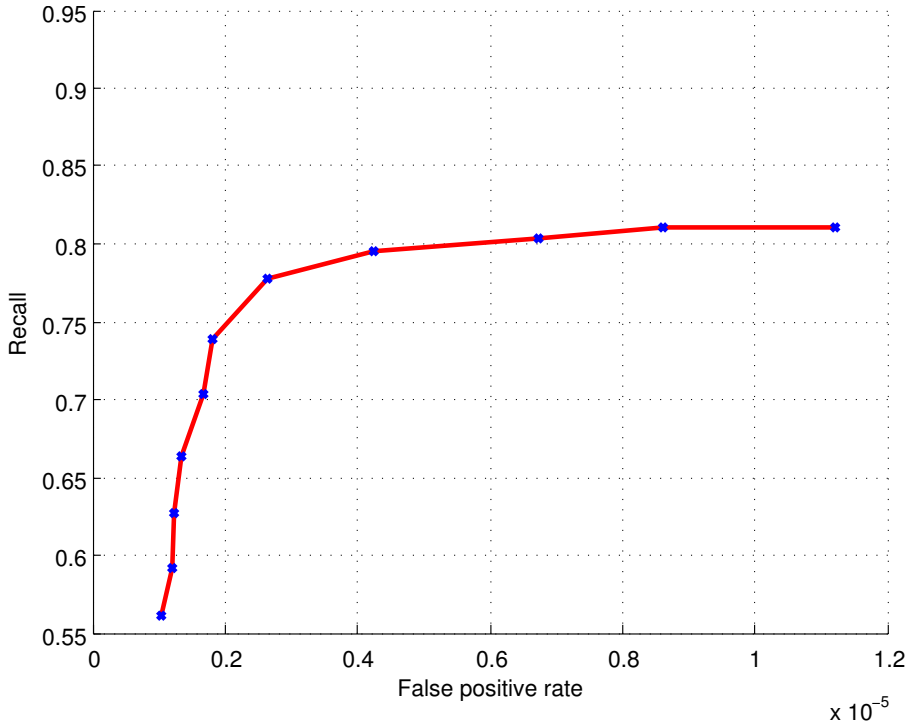


Figure 7.6: *ROC curve of the appearance only classifier generated by sweeping the threshold from* 0.1 *to* 1.1 *with a step size of* 0.1.

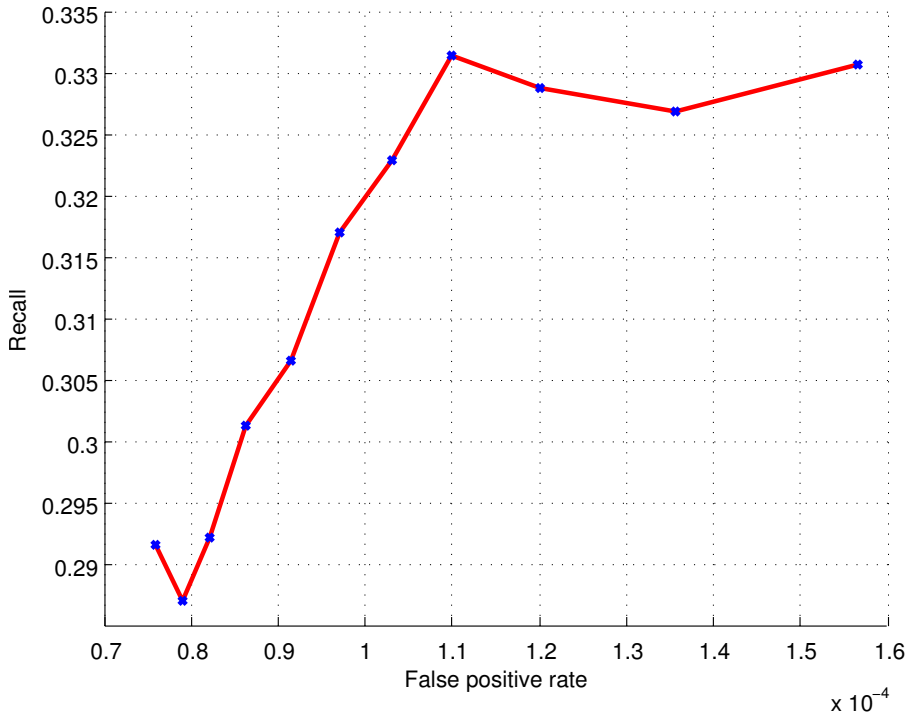


Figure 7.7: *ROC curve of the combined appearance and motion classifier generated by sweeping the threshold from* 0.1 *to* 1.1 *with a step size of* 0.1.

Chapter 8

Animal action recognition

To test the action recognition performance of the second stage of the cascade, a confusion matrix will be generated containing all the tested actions. The matrix show how high a percentage of one action is misclassified as another in the 'action recognition' stage.

The 'action recognition' stage will be tested by running the cascade on consecutive frames in a video containing elephants. All the regions detected as containing an elephant by the 'animal detection and recognition' stage, will be examined by the 'action recognition' stage and classified as one of the two trained actions 'elephant walking left' or 'elephant walking right'.

In order to test the 'action recognition' stage of the cascade without interference from the 'animal detection and recognition' stage, only the detections classified as one of the two tested action will be considered. This exclude detections only detected as an elephant or not detected at all. False positives generated by actions not included in the training set, e.g. a standing elephant recognized as walking, will also not be considered and just ignored.

8.1 Choice of descriptors

The 'action recognition' stage of the classifiers main functionality is to recognize the action performed by the animal detected and recognized by the 'animal detection and recognition' stage of the cascade. The 'action recognition' stage is being tested both with a combination of appearance HOG and motion map HOG descriptors and with motion descriptors only. As has already been mentioned in the descriptor discussion in section 7.1, the motion descriptors are good at encoding the motion pattern performed by the animal and the appearance descriptors are good at encoding the animals texture, shape and color composition. The motion descriptors are a natural choice for an 'action recognition' stage, but by including the appearance descriptor information contained in the pose of the animal combined with the motion descriptors, may improve the performance of the classifier on some actions where motion may be ambiguous.

Each type of the classifiers are tested with appearance and motion map exemplars generated from 5, 10 and 20 averaged consecutive appearance frames and motion maps in order to test if averaging over several appearance frames and motion maps makes the classification step more robust, see section 5.2.5 about multi-motion maps. Classifiers based on single appearance and motion map exemplars will also be tested. This will produce a total of 16 different classifiers, covering two types of actions and eight descriptor combinations. A confusion matrix will be generated for each classifier set-up.

8.2 Setup

The classifier based on motion descriptors and the classifier based on a combined motion and appearance descriptors, will both be using descriptors generated by the HOG algorithm using the same parameters as the 'detector and recognition' stage of the cascade, namely the following parameters:

- Cell size: $8x8$ pixels.
- Block size: $4x4$ cells.
- Linear gradient voting into 18 orientation bins in $0^\circ - 360^\circ$.
- Block spacing stride of 8 pixels.
- $64x64$ detection window.
- Gaussian spatial window, with $\sigma = 8 = 0.5 \cdot block\ width$.

The parameters have been chosen for the same reasons as in section 7.2, but the argument for choosing linear gradient voting into 18 orientation bins in $0^\circ - 360^\circ$ is a bit different. Using 'signed' gradients, i.e. orientations in the interval $0^\circ - 360^\circ$, should help distinguish the two actions 'elephant walking left' and 'elephant walking right', since the most descriptive difference is the color of the motion in the motion maps. Using unsigned gradients will ignore the difference between a dark silhouette and a light silhouette against the grey background, as also described in section 7.2.

8.3 Training and verification data

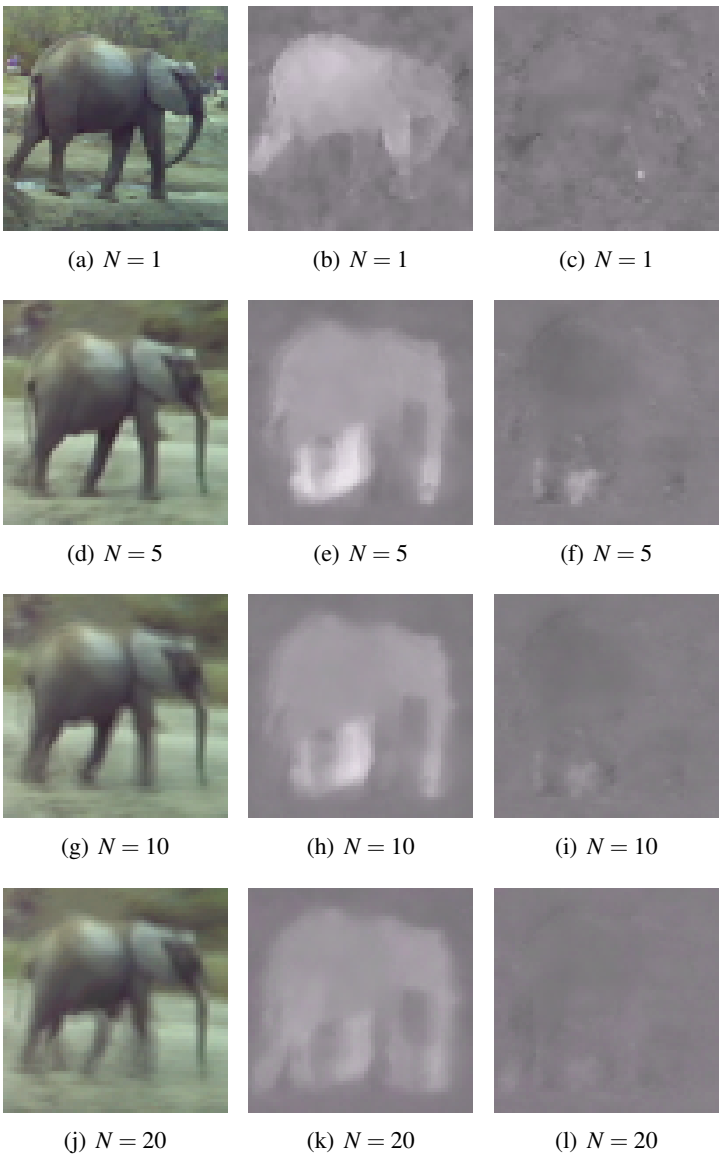
The training data used to train the 'action recognition' classifiers, is from the same set which was used in the 'animal detection and recognition' classifier, see section 7.3. The difference is that only the exemplars where the elephant is moving either to the left or to the right is used. As mentioned above, the average over 5, 10 and 20 appearance and motion map exemplars will be generated for each type of classifiers.

	Appearance / Motion-x / Motion-y			
	$N = 1$	$N = 5$	$N = 10$	$N = 20$
Elephant walking left	715	710	705	695
Elephant walking right	349	344	339	330

Table 8.1: *Positive training exemplars used for training the 16 classifiers.*

Examples of exemplars generated from the appearance frame and motion map and their respective averaged versions are shown in figure 8.1. The training exemplars have been cropped from different sequences of consecutive frames within the frame interval 0 - 4999 in the recorded elephant movie. Selected sequences of frames with elephants in the rest of the elephant movie, in the interval 5000 - 10560 frames, is used to label verification data. From the verification data 446 regions have been labeled containing and elephant walking left and 440 regions containing an elephant walking right. The training exemplars used for training the 'elephant walking left' classifier will be used as negatives for the 'elephant walking right' classifier and vice versa.

The amount of positive exemplars used to train the classifiers are shown in table 8.1.



(a) $N = 1$ (b) $N = 1$ (c) $N = 1$

(d) $N = 5$ (e) $N = 5$ (f) $N = 5$

(g) $N = 10$ (h) $N = 10$ (i) $N = 10$

(j) $N = 20$ (k) $N = 20$ (l) $N = 20$

Figure 8.1: *Exemplars of an elephant walking right. Each row illustrates the exemplars generated by averaging over N subsequent exemplars. Each column illustrates the three different training exemplar types, appearance and motion in the x- and y-direction.*

8.4 Results

As has been discussed in the introduction, confusion matrices will be generated for each type of classifiers in order to test the 'action recognition' stage. This is done to see if the stage is mislabeling one action as another. The first four confusion matrices shown in table 8.2, 8.3, 8.4 and 8.5 are generated with the classifiers based on the motion only descriptors generated from non-averaged, averaged over 5, 10 and 20 exemplars.

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	91%	9%
	Elephant walking right	0%	100%

Table 8.2: *Confusion matrix generated with the motion only classifier trained with the non-average exemplars.*

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	100%	0%
	Elephant walking right	0%	100%

Table 8.3: *Confusion matrix generated with the motion only classifier trained with the average over 5 exemplars.*

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	100%	0%
	Elephant walking right	0%	100%

Table 8.4: *Confusion matrix generated with the motion only classifier trained with the average over 10 exemplars.*

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	100%	0%
	Elephant walking right	0%	100%

Table 8.5: *Confusion matrix generated with the motion only classifier trained with the with the average over 20 exemplars.*

The classifiers based on non-average descriptors, shown in table 8.2, is the only motion only based classifier misclassifying the two actions. It classifies 9% of the 'elephant walking left' ground-truth data as the 'elephant walking right' action, while every single 'elephant walking right' ground-truth data was classified correctly. Using averaged exemplars shows and even better classification performance, which supports this projects hypothesis that using motion information over several frames will help distinguish action. All of the ground-truth 'elephant

walking left' and 'elephant walking right' were classified correctly using the classifiers based on the descriptors made by averaged exemplars.

Even though there is not much to improve on the action recognition performance on this dataset, it has been chosen to test four classifiers based on the combined appearance and motion descriptors generated from non-averaged, averaged over 5, 10 and 20 exemplars. The confusion matrices generated from these classifiers are shown in figure 8.6, 8.7, 8.8 and 8.9.

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	1%	99%
	Elephant walking right	43%	57%

Table 8.6: *Confusion matrix generated with the combined appearance and motion classifier trained with the non-average exemplars.*

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	28%	72%
	Elephant walking right	80%	20%

Table 8.7: *Confusion matrix generated with the combined appearance and motion classifier trained with the average over 5 exemplars.*

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	54%	46%
	Elephant walking right	90%	10%

Table 8.8: *Confusion matrix generated with the combined appearance and motion classifier trained with the average over 10 exemplars.*

		Detected	
		Elephant walking left	Elephant walking right
Actual	Elephant walking left	85%	15%
	Elephant walking right	94%	6%

Table 8.9: *Confusion matrix generated with the combined appearance and motion classifier trained with the with the average over 20 exemplars.*

These results are somewhat peculiar. When using the classifier based on the non-average descriptors, the detections seem to favor the 'elephant walking right', where 99% of the 'elephant walking left' ground-truth labels have been detected as 'elephant walking right'. Increasing the amount of exemplars used in the averaged descriptors, seems to increase the amount of detection hits being classified as the 'elephant walking left' action. An explanation to this behavior might be found by the uneven amount of training data used for the two different actions. Including

appearance data in the classifier seem to outweight the motion data so heavily that the SVM can not use the motion data to correctly separate the two classes.

Chapter 9

Human recognition

Due to the low amount of actions in our own dataset the verification of the 'action recognition' stage of the classifier has not been tested against enough different action types in order to prove the action recognition efficiency. In order to verify the 'action recognition' stage the stage is tested against the KTH-dataset [45]. The dataset contains the actions 'handclapping', 'handwaving', 'boxing', 'walking', 'running' and 'jogging'. The results will be presented in confusion matrices as in the verification of the elephant, chapter 8.

The 'action recognition' stage will be tested against the human candidate regions resulting from the 'detector and recognition stage', which has been trained on human exemplars cropped from the KTH-dataset. Since the human detection part is not in the scope of this report, the detection performance of this classifier will not be discussed.

9.1 Choice of descriptors

Since the combined appearance and motion descriptors proved to have very bad performance used in an 'action recognition' stage in chapter 8, it has been chosen only to test the motion only method.

9.2 Setup

The setup is almost identically to the one used in chapter 8, with only two deviations. The first difference is the use of $48x48$ detection windows instead of $64x64$, due to the smaller videos included in the KTH-dataset ($160x120$). The $48x48$ window size was chosen as it cover the smallest occurrences of humans in the dataset. The other deviation is the use of 'unsigned' gradients instead of 'signed' gradients. The 'unsigned' gradients are used to eliminate the direction dependency when using the motion map exemplars, e.g. the purpose is only to recognize if a human is walking and not if the human is walking left or right with respect to the camera. This was chosen as other action recognition algorithms tested against the KTH-dataset do not distinguish

direction. The 'signed' vs 'unsigned' gradient behavior is explained in chapter 7 under the setup of the combined appearance and motion classifier.

9.3 Training and verification data

The training data used for the human action recognition classifier consists of motion map exemplars cropped from the motion maps generated from the videos in the KTH-dataset. The KTH-dataset provides six different action sets 'handclapping', 'handwaving', 'boxing', 'walking', 'running' and 'jogging', performed under four different conditions and includes data for 25 different persons. The data for the different persons are split into two parts, one training part consisting of the eight different persons, named person 11, 12, 13, 14, 15, 16, 17 and 18 and one verification part consisting of the eight different persons named person 19, 20, 21, 23, 24, 25, 01 and 04.
Some actions from the KTH data set contain actions that can have different orientations, such as 'boxing', 'walking', 'jogging' and 'running'. for 'boxing' the verification set contains 'boxing' sequences both to the left and to the right. The training set only contain two videos of a person 'boxing' to the right, which seems quite low to learn useful features. Since we are using unsigned gradients in the HOG descriptor, for this test, we have mirrored all our training image pairs and used those for training also. This increases our training exemplars for opposing directions.

Examples of the cropped exemplars used for the motion descriptor based classifier, is shown in figure 9.1.

The number of positives and negatives used for the training used, is shown in table 9.1.

The small amount of 'jogging', 'walking' and 'running' exemplars used to train the classifiers are due to the fact, that for these actions, the people pass in and out of the images. The people in the videos are fairly close to the camera, leaving very few frames where a square region of the person can be extracted. This have a great impact on especially the action 'running', where it was not possible to generate exemplars averaged over 20 exemplars, since the person was not inside the extractable area for 20 frames in most sequences.

	Motion-x / Motion-y			
	$N = 1$	$N = 5$	$N = 10$	$N = 20$
Handclapping	3010	2860	2719	2410
Handwaving	3266	3106	2946	2626
Boxing	3264	3104	2944	2624
Jogging	1320	1000	680	50
Running	762	482	202	N/A
Walking	2034	1724	1414	808

Table 9.1: *Positive training exemplars used for training the six different action classifiers.*

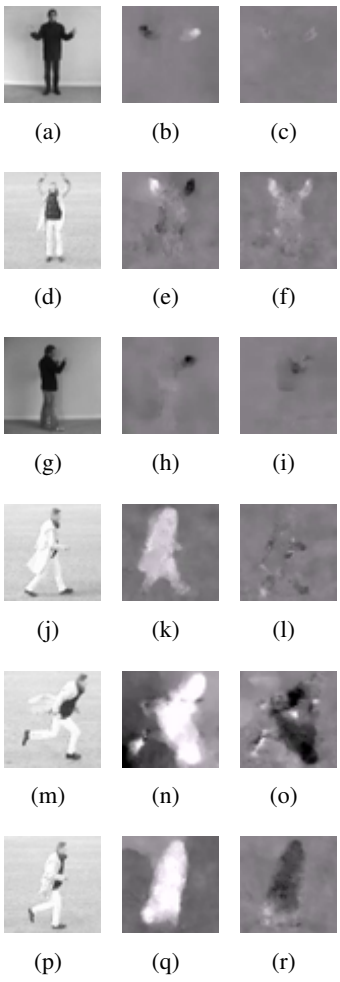


Figure 9.1: *The first column shows the appearance exemplar corresponding to the motion map exemplars shown in the second and third column. Note that only the second and third column are used for training the 'action recognition' classifier. The rows from top to down represents 'handclapping', 'handwaving', 'boxing', 'walking', 'running' and 'jogging'.*

9.4 Results

As has been discussed in the introduction, confusion matrices will be generated for each type of classifier in order to test the human 'action recognition' stage. This is done to see if the stage is mislabeling one action as another. Table 9.2 shows the results from using non-average exemplars, table 9.3, 9.4 and 9.5 show the results from averaging over 5, 10 and 20 exemplars respectively.

		Detected					
		Boxing	Handclapping	Handwaving	Jogging	Running	Walking
Actual	Boxing	16%	36%	48%	0%	0%	0%
	Handclapping	7%	45%	47%	0%	1%	0%
	Handwaving	5%	33%	62%	0%	0%	0%
	Jogging	7%	4%	43%	41%	5%	0%
	Running	7%	5%	47%	33%	8%	0%
	Walking	8%	11%	63%	17%	0%	0%

Table 9.2: *Confusion matrix generated with the motion only classifier trained with the non-average exemplars.*

It is easy to see in table 9.2, that a great number of actions were mislabeled. Especially the action 'handwaving' seems to dominate the other actions in all of the action sequences. Another thing to notice, is the great confusion between the actions 'boxing', 'handclapping' and 'handwaving'. By looking at the non-averaged motion maps for the actions, we see that the actions look very similar. The 'boxing' sequences may very well be confused with the 'handclapping' and 'handwaving' sequences since both left and right boxing characters were trained into the same SVM model. Since most of the 'boxing' motion occupy the same areas of the image as 'handclapping' and 'handwaving' and both left and right were trained into the same model, there is a great chance that the classifier could not build a good set of negative weights for the set.
The confusion between 'walking', 'jogging' and 'running' is not very surprising, as our algorithm does not recognize speed very well. As the poses look very similar, it is hard to distinguish the three actions in low resolution videos, especially from motion. Looking at fig. 9.2 it is also clear that the motion data from the 'walking', 'jogging' and 'running' actions are very noisy, which help further explain the problems seen.

(a) Walking (b) Running (c) Jogging

Figure 9.2: *Examples of noisy 'Walking', 'Running' and 'Jogging motion maps. Left images are X and right images are Y motion maps.*

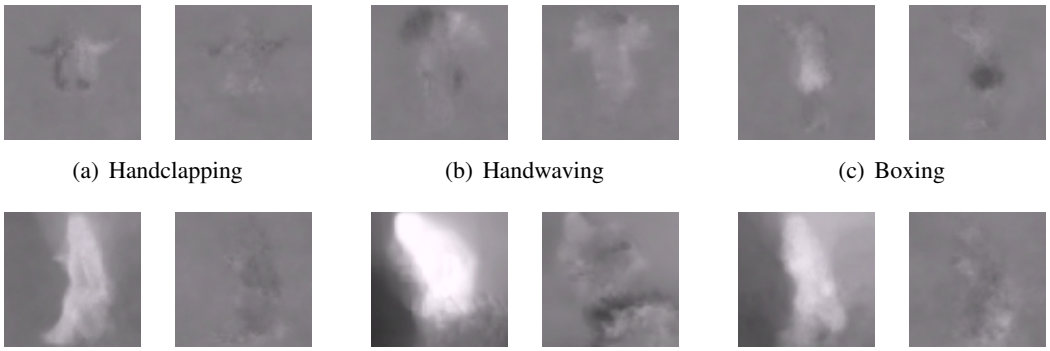
		Detected					
		Boxing	Handclapping	Handwaving	Jogging	Running	Walking
Actual	Boxing	23%	53%	24%	0%	0%	0%
	Handclapping	14%	55%	30%	0%	1%	0%
	Handwaving	17%	27%	53%	0%	2%	0%
	Jogging	9%	15%	16%	47%	11%	2%
	Running	15%	15%	18%	34%	16%	1%
	Walking	18%	19%	27%	26%	6%	4%

Table 9.3: *Confusion matrix generated with the motion only classifier trained with the average over 5 exemplars.*

		Detected					
		Boxing	Handclapping	Handwaving	Jogging	Running	Walking
Actual	Boxing	26%	53%	20%	0%	1%	0%
	Handclapping	16%	58%	25%	0%	1%	0%
	Handwaving	17%	19%	60%	0%	4%	0%
	Jogging	11%	7%	34%	14%	23%	11%
	Running	13%	10%	29%	14%	24%	10%
	Walking	16%	7%	52%	2%	7%	17%

Table 9.4: *Confusion matrix generated with the motion only classifier trained with the average over 10 exemplars.*

Increasing the number of average frames decrease the confusion between 'handclapping' and 'handwaving', as a more clear picture of the dominant movements appear in the motion maps. The figure also shows that when using more average frames, the motions start to fade, as opposing motion cancel each other out in the average images. The 'walking', 'jogging' and 'running' actions are still well separated from the first three actions, but the internal confusion is still great. The increased amount of images being averaged, only help eliminate the noise seen in the non-averaged images, but does not give information as to how to distinguish them. Examples of the see fig. 9.3 for examples of the N10 motion maps.



(a) Handclapping (b) Handwaving (c) Boxing

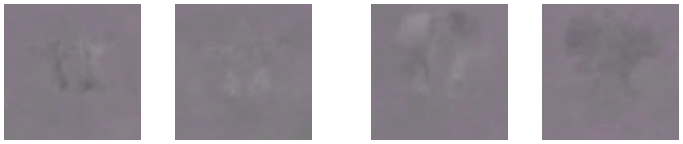
(d) Walking (e) Running (f) Jogging

Figure 9.3: *'Handclapping', 'Handwaving', 'Boxing', 'Walking', 'Running' and 'Jogging' motion maps from the N10 training set. Left images are X and right images are Y motion maps.*

		Detected					
		Boxing	Handclapping	Handwaving	Jogging	Running	Walking
Actual	Boxing	46%	50%	3%	0%	0%	0%
	Handclapping	21%	76%	2%	0%	0%	0%
	Handwaving	28%	34%	35%	0%	1%	2%
	Jogging	17%	34%	7%	0%	2%	41%
	Running	31%	34%	5%	0%	12%	19%
	Walking	23%	41%	6%	0%	1%	29%

Table 9.5: *Confusion matrix generated with the motion only classifier trained with the average over 20 exemplars.*

As the number of frames in the average is increased to 20, the 'handclapping' recognition rate dramatically increase. Looking at the average frames for the N20 averages, fig. 9.4, it becomes apparent that the images hardly contain any distinguishable data for 'handclapping' and 'handwaving'. It is hard to tell if the SVM classifier may be able to extract useful information from the HOG descriptors for such data, but it seems unlikely as many 'walking' frames are also suddenly classified as 'handclapping'.



(a) Handclapping (b) Handwaving

Figure 9.4: *'handclapping' and 'handwaving' examples from the N20 training set. Left images are X and right images are Y motion maps.*

Chapter 10

Discussion

The performance of the 'animal detection and recognition' stage performs very well on our elephant class. The detection rate is approximately 4%-9% lower at a false positive rate of $4x10^{-6}$ than what Dalal and Triggs[12] got from their human detector running on the INRIA dataset. Dalal and Triggs trained their classifier to find humans seen from a side view at any of the 360° orientations, since the human body have approximately the same shape from any of these orientations. Our elephant classifier is also trained on a similar 360° orientation scheme, even though elephants do not generalize well to a vertically symmetric model, so a detection rate decrease of 4%-9% compared to the human detector does not seem poor.

The results of the 'action recognition' stage on the elephant dataset, presented in the confusion matrices in fig. 8.3 through 8.5 did seem impressive, with a 100% recognition rate using 5 averaged frames and up. Using non-averaged frames did not do as well as the averaged frames, which indicates that using averaged frames does in fact help the action recognition with respect to an elephant walking. The results also seem to indicate that increasing the amount of frames used for the average, improves the results. As mentioned in section 5.2.5, this may only be true to some extend. As the number of frames per average is increased, the response time of the classifier is decreased. This means that an animal walking left, may be walking left for several seconds, before an appropriate average frame is generated for the classifier to recognize, or if the animal changes from one action to another the correct action is not detected until the average has been completely replaced by the motion of the new action. The amount of frames used in the average is the determining factor of how long these transition periods are, as the worst case transition time can be calculated as:

$$T = \frac{N_{avg}}{FPS_{vid}} \, [s],$$

where N_{avg} is the number of frames being averaged and FPS_{vid} is the frames pr. seconds in the corresponding videofeed. If the action to be detected can be performed within the time T, the average may often not be able to converge fast enough for the detector to be able to detect the action. It is apparent from the above equation, that the algorithm is not invariant to the frame rate of the camera or video. It is therefor very important to choose an average length that makes sense

for the action being performed and the equipment used for observing the scene. Our framework is developed so that a threaded process samples the video input at a predetermined frame rate, compensating for variable frame rates from the input.

Unfortunately, testing only the two different actions 'elephant walking left' and 'elephant walking right' against each other, is not enough to validate the 'action recognition' stage. This led to the test of our system against a public available dataset named KTH, created by Christian Schüldt et al. [45].

The results shown as confusion matrices in table 9.2 to 9.5, did show some very poor results on the KTH dataset. One of the things to notice from the results are the great confusion between the actions 'boxing', handclapping' and 'handwaving' when using non-average exemplars. This may be due to the very similar motion map training exemplars seen between them. The 'boxing' might very well be confused with the 'handclapping' and "handwaving' actions, since both left and right boxing exemplars were trained into the same SVM classifier. In order to avoid some of the confusion between the action 'boxing' and the actions 'handclapping' and 'handwaving' one could train two classifiers for 'boxing', one for each direction.
Increasing the amount of exemplars used for averaging appear to help distinguishing the actions 'boxing', handclapping' and 'handwaving', even though the training images seem to fade out as opposing motions cancel each other out. This could indicate, that a different approach of storing the pixel movement information in the motion maps is needed. This could instead be done by storing the amplitude of the motion vector instead of using its x and y components or by storing the positive and negative values of the x and y components in separate color channels.

There were also found to be a great amount of confusion between the actions 'jogging', 'running' and 'walking', probably due to the noisy motion map exemplars generated with the optical flow algorithm and the fact that our approach does not generalize speed very well. This confusion is not changing much when increasing the amount of frames used for averaging. In order to improve these results, one should probably use higher resolutions for the videos in order to get better results from the optical flow algorithm.

Comparing the results with the approach of using local descriptors together with SVM, by Christian Schuldt et. al. [45], on the same KTH dataset, shows some similarities to our results. This approach does also confuse between the similar actions 'walking', 'jogging' and 'running' and the other group of similar actions 'boxing', 'handclapping' and 'handwaving'. Even though they have similar confusion between the actions as our system, they do have an overall better recognition rate per actions. Another approach using local descriptors for spatio-temporal recognition, by Ivan Laptev et. al [47], shows impressive results, with an almost perfect detection rate in all of the six actions. It is important to note that these methods uses local image descriptors and to the best of our knowledge does not handle frames with more than one subject. We are concidering the whole frame supporting multible detections, which makes our problem harder. However combining our global HOG approach in the first cascade, with a classifier using local descriptors in the second stage might improve the action recognition significantly, and at the same time

support multible detections.

PART IV

CONCLUSION AND FUTURE WORK

Chapter 11

Conclusion

The goal of this project is to develop a proof-of-concept system that is able to detect an animal, recognize the action being performed and augment the information onto a screen for a user to see while walking through a zoo or driving through a animal park. The system is designed to be used with a head-mounted-display with front mounted cameras for input and internal screens for display of the results. The HMD setup requires the implementation to be robust to camera movements, lighting conditions and be able to perform the tasks online.
To built a system living up to the above stated requirements we have designed a two-stage cascaded binary linear SVM-classifier trained on HOG descriptors. The HOG descriptor provides robustness to variation in lighting and creates dense descriptors using gradients to encode the texture, shape and color composition of the animals. The first cascade stage is the animal detector stage and the second stage is the action detection stage.
We found that using appearance images only, with signed gradients arranged into 18 bins for the 'animal detection and recognition' stage gave the best results. The detection rate was approximately 80% at a false positive rate of approximately $4x10^{-6}$ false positives on our own verification set. The detector is trained so that it can detect elephants from a 360° side view, even though elephants do not have a vertical symmetry that generalize as well as humans.
For the 'action recognition' stage we found that using motion maps only, performed much better than using combined appearance and motion features. We again used signed gradients arranged into 18 bins, to be able to differentiate between left versus right motion, as well as upwards versus downwards motion. We also found that for recognizing an elephant walking left versus walking right using average images over 5 frames or more eliminated any confusion of the two actions. This lead us to test the classifier on the KTH action dataset, to see how the action detection performed on slightly more complex and larger action sets. The results on the KTH-dataset were not satisfactory, and part of the reason was the low resolution of the dataset, which resulted in corrupted motion data. The results also revealed that a list of improvements are needed to improve our action recognition method, as the main reason for the bad results are motions canceling each other out when average frames were constructed.
We found that compensating the motion maps with the average image motion eliminates camera motion in the motion maps in most cases, for low to medium camera movements. We also found that using the GPU implementation of the optical flow algorithm and the ODLToolkit HOG im-

plementation is not able to compute the needed data in an online application at a resolution of 640x480 with the configurations used in this report.

Chapter 12

Future Work

There are implementations of the HOG algorithm and settings for the optical flow algorithm that may be able to increase the computational performance to a point where online performance is achievable. A method of building a detector from a cascade of rejectors, based on HOG features already exist and claims to be able to detect humans online[14]. Implementing the HOG descriptor on a GPU may also speed up the descriptor enough to allow online detection, when used in an SVM.

In our implementation, the ODLToolkit library is using global settings for the HOG objects, as it was not designed for multi-class detection. Implementation of multi HOG support in the ODLToolkit library will allow different binning settings and different SVM thresholds for each cascade stage, may also allow for much better fine tuning of the cascades.

Action detection performance can possibly be further improved by implementing a tracking method such as Kalman filtering, to keep track of the animals detected even when the detector may fail on single frames. Since the 'action recognition' stage rely heavily on the 'animal detection and recognition' stage, the implementation of a tracker in the first stage may very well greatly improve the robustness of the second stage. The 'action recognition' stage may also be improved by implementing a voting system that can allow more action classifiers to vote on an exemplar. Such a voting system can be used to both keep a detection stable, when outliers appear but also allow the classifier to have multiple possible classes allowing fast switching if strong evidence for nearby class appear, when a weak classification is found.

A significant weakness of our 'action recognition' stage of our cascade is the bad recognition performance to movements such as hand waving using larger numbers of exemplars in the average, as the back and forth movements start to cancel out. For actions where the 'sign' of the movement is not important, using 'unsigned' flow frames could possibly eliminate the problem. A different approach could be to store negative and positive motions in separate motion maps.

The implemented method for compensating for camera motion is sensitive to extreme outliers in the motion data and large foreground movement. A different method for compensating camera motion could possibly eliminate these error factors, such as determining the movement from a median or combined median and mean filter, meaning over N number of neighbors of the median. A more robust method could be the utilization of the two cameras in the HMD to retrieve stereo depth, in order to determine which pixels are actual background pixels.

With the development of less computational intensive algorithms and exploiting the OpenGL framework can lead to the ability to implement the system on modern mobile devices, such as smart phones and tablets, making the system much more appealing to the users. In this project the focus has been on the implementation of the detection and recognition problem and left the augmentation perspective mostly unexplored.

PART V

APPENDIX

Appendix A

Performance Optimization of HumanEva using OpenGL

During our stay at CMU and Disney Research Pittsburgh we were asked to help out on a project that our supervisor Leonid Sigal has been working on, see [48]. We were asked to recode parts of a MATLAB function named `propagate` into C-code, as to improve computation time. The HumanEva project aims to estimate the pose of a person, using an annealing particle filter, where as the `propagate` function is responsible for calculating the likelihood for several hundreds of particles for each layer for each pose estimation.

Each particle in the particle filter generate a skeletal pose, within a limited space of joint poses. Each skeleton is then skinned with a predefined 3D model, which is then projected into a 2D model. The projection is made for each view the subject has been recorded from with all the relevant distortion, rotation and translation parameters. The 2D projection is then matched against a background subtracted image obtained from the view corresponding to the 2D projection. The likelihood of a guessed pose being the actual pose observed is then calculated based on the overlapping and non-overlapping regions of the 2D projections and the background subtractions, is then calculated for each particle.

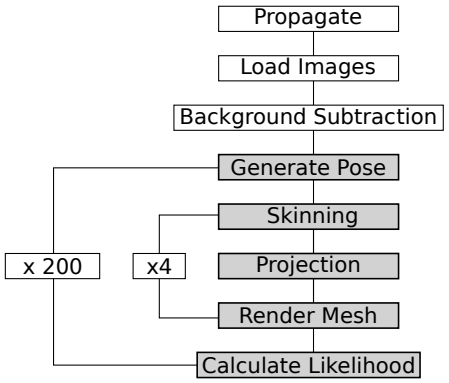


Figure A.1: *Simplified structure of the MATLAB code before optimizations. Grey boxes indicate functions that will be replaced.*

A simplified code structure can be seen in fig. A.1. The gray boxes indicates MATLAB code which will be replaced by either C or OpenGL code. In fig. A.2 the simplified structure of the optimized code can be seen. The bright blue boxes indicate C-code and the bright red boxes indicate OpenGL code.

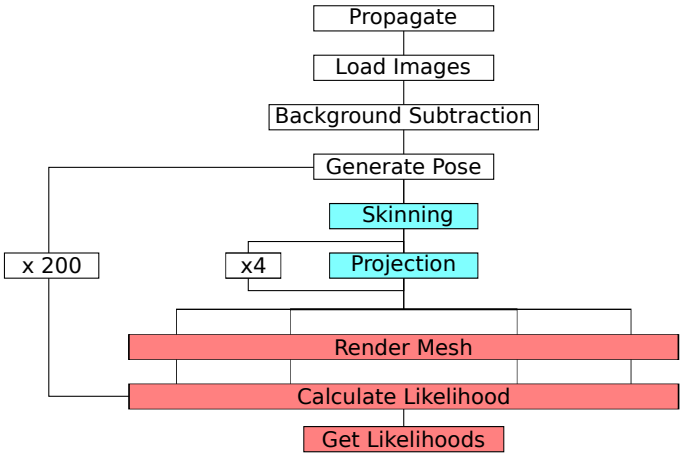


Figure A.2: *Simplified structure of the optimized code. Bright blue boxes are MATLAB code replaced by C-code and the red are replaced by OpenGL code.*

Since the code was all written in MATLAB and we were only asked to optimize some parts, it was necessary to interface the MATLAB code with C-code. This was done using the MEX libraries in MATLAB allowing data transfers between MATLAB and standard C-code [49].

To speed up the code we decided to use OpenGL[40] in combination with shaders written in the shader programming language GLSL[50] to create fast parallel functions. Not every function is feasible to do in OpenGL, whereas they just were be implemented in C-code.

To outline the changes that has been made in the MATLAB code, the following sections will explain the old MATLAB code and where it needs optimizations. Note that throughout this text function names will be written as `function_name` and `m_` in front of a function's name will indicate a not yet optimized function, a `c_` will indicate an optimized function in C-code and a `gl_` will indicate an optimized function which is implemented using OpenGL and GLSL.

A.1 MATLAB code before optimization

In order to change anything in the `propagate` function, it's functionality needs to be known. To simplify the illustration of the `propagate` function's functionality, it has been chosen to split it up in different functions, i.e. `m_image_initialization`, `m_skinning`, `m_2D_projection`, `m_mesh_rendering` and `m_likelihood`.

For the sake of the overview, the above functions will shortly be introduced leaving out the details. The details of the optimizations done to improve the functions are explained in chapter B.

`m_image_initialization`
This function takes care of the initialization of all the images used on the `propagate` function. This includes the calculations of the background subtracted images.

`m_skinning`
Each particle in the particle-filter is a predicted pose, which is initially just represented as a skeleton. The skeleton needs to be skinned in order to be compared to a background subtracted image. The skinning is basically made out of various matrix and vector multiplications, where each vector or matrix represents the skin model, weights and skeleton model. The output from the `m_skinning` function is a 3D mesh of triangles.

`m_2D_projection`
The 3D mesh of a pose guess needs to be projected down to a 2D image which is comparable with a background subtracted image. This is done in `m_2D_projection`, where the 3D coordinates will be projected into a 2D coordinate image plane given the four different camera views and intrinsic and extrinsic camera parameters.

`m_mesh_rendering`
This function rasterizes the 2D image plane of the pose guess into an image.

`m_likelihood`
The `m_likelihood` function calculates the likelihood between the guessed pose image and a background subtracted image. The likelihood for the different views is averaged and returned.

Pseudo code describing the `propagate` function is shown in figure A.3.

```
propagate() {
  m_image_initialization();
  for m = 1:5   // num_layers     {
    for p = 1:200   // num_particles     {
      m_skinning();
      for v = 1:4  // num_views    {
        m_2D_projection();
        m_mesh_rendering();
      }
      m_likelihood();
    }
  }
}
```

Figure A.3: *Pseudo code illustrating roughly where the different functions are called and shows how many times they are called.*

As can be seen in the pseudo code, some of the functions are called hundreds of times to compute the 5 layers in the annealing particle filter. If any of these functions are slow, it will have a big impact of the overall speed performance of the `propagate` function. To see if any of the above functions need optimization, profiling will be done of the MATLAB code in the next section.

A.2 Timing results before optimization

To be able to do any optimization, the MATLAB code should be benchmarked in order to see where the code needs improvement. The function which needs optimizations in the HumanEva project is found to be the `propagate` function of the annealing particle filter. The profiling results which prove the `propagate` function to be the dominant one is not included because we were only asked to optimize the `propagate` function.

The timing results for the functions called from within the `propagate` function are shown in table A.1 and A.2. It has been chosen to run two rounds of pose estimation, which will lead to two calls of the `propagate` function. The timings are found on an Intel(R) Core(TM) 2 Duo P8700@2.53GHz with an ATI Radeon 4650, running MATLAB 2010b on Ubuntu 10.10 64-bit. The timings were found by using the 'tic - toc' approach [51] instead of the build in profiler in MATLAB. The profiler was not used due to its self-time and overhead profiling MATLAB code. The extra time the profiler would take profiling the MATLAB code, would lead to wrong results of the timing results of the OpenGL functions, due to the parallel nature of the OpenGL backend. The OpenGL functions would simply have time to finish in the background before it would be called again, leading to an almost zero timing result.

Layer	#5	#4	#3	#2	#1
`m_image_initialization` [sec]	0.51	0.00	0.00	0.00	0.00
`m_skinning` [sec]	15.10	14.81	14.99	14.97	15.12
`m_2D_projection` [sec]	27.86	27.33	27.25	28.22	29.65
`m_mesh_rendering` [sec]	51.97	50.55	51.36	50.55	51.36
`m_likelihood` [sec]	17.16	16.84	16.92	17.63	17.66
Misc MATLAB code [sec]	2.40	2.47	1.48	3.63	4.21
Total pr. layer [sec]	115.00	112.00	112.00	115.00	118.00
Total pr. pose [sec]					572.00

Table A.1: *Timing results for the MATLAB functions over each layer for the first pose. For each layer the functions process 200 particles for each of the four camera views.*

Layer	#5	#4	#3	#2	#1
`m_image_initialization` [sec]	0.50	0.00	0.00	0.00	0.00
`m_skinning` [sec]	14.90	15.09	15.10	15.03	14.98
`m_2D_projection` [sec]	27.97	28.92	29.01	28.53	28.78
`m_mesh_rendering` [sec]	52.91	52.11	52.12	51.17	51.55
`m_likelihood` [sec]	17.28	17.76	17.59	17.25	17.46
Misc MATLAB code [sec]	4.44	3.12	2.18	2.02	2.23
Total pr. layer [sec]	118.00	117.00	116.00	114.00	115.00
Total pr. pose [sec]					580.00

Table A.2: *Timing results for the MATLAB functions over each layer for the second pose. For each layer the functions process 200 particles for each of the four camera views.*

As can be seen in the timing-results for the two pose computations, the total computation time per pose is relatively high when the pose has to be calculated for each frame in a video clip.

To trace the functions needed to be optimized, the average performance of the different functions should be considered. The graph A.4 shows the average computation time for the different MATLAB functions over two rounds of pose estimations.

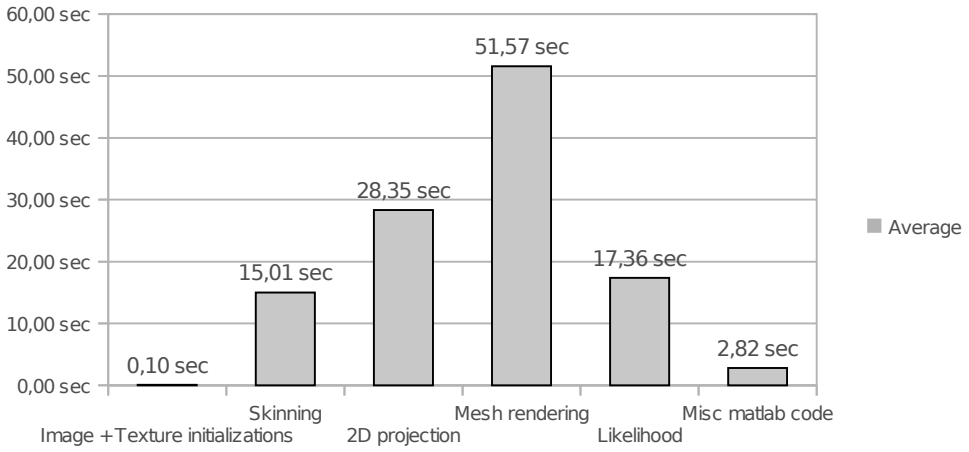


Figure A.4: *Average performance for the MATLAB functions computing two poses with five layers in each.*

As can be seen in the above graph, the functions `m_skinning`, `m_2D_projection`, `m_mesh_rendering` and `m_likelihood` are the predominant reason for the slow computation time. If these could be optimized, the `propagate` function in the HumanEva project should see a performance improvement.

In the next chapter will functions, which addresses these performance issues, be proposed.

Appendix B

Implementation of optimized functions

B.1 Mesh rendering

To compare a pose guess from the particle filter to the background subtracted image, it is necessary to render the pose guess to a texture that can be stored on the GPU. To do this, the initial mesh pose is extracted from MOCAP data and fed to the particle filter. The particle filter then generates a set of randomized guesses, distributed around the initial pose by a Gaussian function and constrained by physical constraints of the human body. The resulting poses guesses are then skinned using the skinning function described in section B.3.1, resulting in an array of vertices in a polygonal mesh. The resulting 2D mesh is scaled and compatible with OpenGL's drawing functions.

Since the mesh is in one large and ordered array, rendering the mesh using OpenGL's Vertex Array support is a straight forward procedure and allows for high speed rendering. The vertex array method allows transfer of the whole mesh before rendering anything to the screen, limiting the amount of CPU to GPU data transfers compared to immediate mode rendering [52].

The rendering of the mesh was the largest slowdown in the MATLAB code as can be seen in figure A.4, since software rendering has always been a hard task for the CPU. By moving this operation to the GPU using OpenGL, the rendering process is transfered to a combination of framework and hardware built for the sole purpose of this type of operations and should provide significant improvements. The rendering function, `gl_render_mesh` is implemented as seen below.

```
gl_render_mesh(gl_context, dims, vertice_index, vertice Array, [optional arguments])
Optional arguments: ( 'Color', float[R,G,B] ) , ( 'Linewidth',float ) , ( 'Texture', texture )
```

The `gl_render_mesh` function is a multipurpose function that can render a mesh either to a texture or to the screen. Rendering to a texture is done by giving the optional string `'Texture'` followed by `texture` which is an integer holding the OpenGL texture ID of the texture in which to store the output. The other two optional input parameters, `'Color'` and `'Linewidth'` allows for change of mesh color and rendering as wire frame. Setting `'Linewidth'` to 0.0 disables wire frame rendering. Both `'Color'` and `'Linewidth'` should only be set when rendering to screen, since neither wire frame or colored mesh textures are supported by the other parts of the code

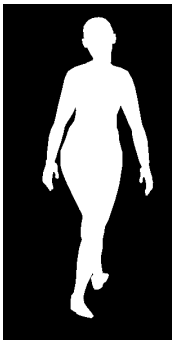


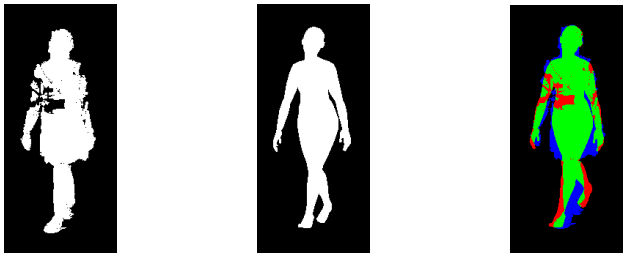
Figure B.1: *Mesh rendered using the OpenGL rendering function.*

used for likelihood calculations. In figure B.1 an example of a mesh render is shown.

B.2 Likelihood

As has been found in section A.2, the `m_likelihood` function needs optimizations which should lead to an overall performance improvement of the MATLAB function `propagate`. In order to illustrate what have been done to improve the `m_likelihood` function, it has been chosen first to highlight how the `m_likelihood` function works.

The purpose of the `m_likelihood` function, is to give an estimation of how well a background subtracted image fits the guessed pose, i.e. a particle. In the case of the current MATLAB code, the fit of a subtracted image and the guessed pose is found as how much the silhouettes from the images overlap each other versus the penalty of non-overlapping regions for both silhouettes [48]. An illustration of the overlapping and non-overlapping regions can be seen in figure B.2.



(a) *Binary background subtracted image M^b.*

(b) *Binary image of the silhouette representing the guessed pose M^f.*

(c) *Illustration of the regions R_t,B_t and Y_t.*

Figure B.2: *Visualization of how MATLAB and OpenGL expects to read images from memory.*

The idea is to minimize the non-overlapping regions illustrated as the colors red and blue, therefore maximizing the overlapping region illustrated with a green color.

The sum of the pixels p for each region is found as:

$$R_t = \sum_p (M^f(p)(1 - M^b(p))) \tag{B.1}$$

$$B_t = \sum_p (M^b(p)(1 - M^f(p))) \tag{B.2}$$

$$Y_t = \sum_p (M^f(p)M^b(p)) \tag{B.3}$$

The likelihood can then be calculated as:

$$-log p(y_t|x_t)^v \propto (1-\alpha)\frac{B_t}{B_t + Y_t} + \alpha\frac{R_t}{R_t + Y_t}$$

$p(y_t|x_t)^v$ is the probability of the image observation conditioned on the pose estimate for the view v and particle t. For more information, see [48]. This likelihood should be evaluated as the

average likelihood over all the four views.

$$-logp(y_t|x_t) = \frac{1}{4}\sum_{v} -logp(y_t|x_t)^v$$

This likelihood will be returned by the `m_likelihood` function for further processing.

To compute the likelihood for 200 particles sequentially on the CPU, does obviously consume a lot of computation-time even with small images. To address this, it has been chosen to calculate the likelihood on the GPU using OpenGL as has been done in the `gl_render_mesh` function described in section B.1. This will also make use of the already rendered mesh texture (pose guess) generated by the `gl_render_mesh` function. Keeping the textures in the video-memory without having to download the texture to the system-memory, should speed things up due to a well known small memory download bandwidth from the video-memory to the system-memory.

To write a GPU implementation of the `m_likelihood` function, it has been chosen to split the function up in two parts to make it easier utilizing different shaders. The first part will generate a silhouette-image like the one illustrated in figure B.2(c). The second part will summarize over the different regions and calculate and place the likelihoods in a texture depending on the current evaluated particle. This texture will from now on be referenced as the `probability_texture` and will have the size one times the number of particles. The later is done to limit the download of textures from the video RAM to the system RAM, by only downloading the `probability_texture` to the system RAM after all the, in this case 200, probabilities have been calculated.

The two separate functions will from now be called `Silhouette` and `Average` and their OpenGL / GLSL implementation will be explained in the section B.2.1 and B.2.2. It should be noted that `Silhouette` and `Average` are internal functions in the resulting mex file and will not be exposed to the MATLAB interface. The likelihood function which is exposed to the MATLAB interface `gl_likelihood` is implemented as seen below.

```
gl_likelihood(gl_context, textures_background_subtraction, textures_mesh, texture_probability,
              textures_silhouette, num_particle,num_views);
```

Where `num_views` is the number of views to compute the likelihood for. `textures_background_subtraction` is an array containing `num_views` background subtracted textures. `textures_mesh` is an array containing `num_views` pose guesses. `texture_probability` is an OpenGL texture number, in which one wants to save the probabilities. `textures_silhouette` is an array of `num_views` empty textures, which the `gl_likelihood` function will use for temporary silhouette calculations. `num_particle` is the current computed particle. `num_views` is simply the number of views.

B.2.1 Silhouette

To calculate the likelihood of a certain pose, guessed by the particle filter, we must calculate the differences and overlaps of the background subtraction result and the result of the pose guess. To do this a shader is designed to do GPU accelerated comparisons of the two images, which are stored in textures on the graphics card, from here on called `silhouette_texture`.

The shader is designed so that it will utilize the three color channels of the textures. The red color channel is used to draw the unaccounted for pixels of the pose guess, the blue channel is used for the similarly unaccounted for pixels of the background subtraction result and the green channel is used to store the intersection of the two textures. These values represent the inner calculations of the sums of R_t, B_t and Y_t in equations B.1, B.2 and B.3. A pixel in the `silhouette_texture`, $M^s(p)$, is calculated as:

$$R(p) = M^f(p)(1 - M^b(p)) \tag{B.4}$$

$$B(p) = M^b(p)(1 - M^f(p)) \tag{B.5}$$

$$G(p) = M^f(p)M^b(p) \tag{B.6}$$

$$M^s(p) = \begin{pmatrix} R(p) \\ G(p) \\ B(p) \end{pmatrix} \tag{B.7}$$

$M^s(p)$ is a three channel RGB pixel in the `silhouette_texture` M^s.

The calculations are done in a shader, which utilize the multiple GPU cores on the graphics card to calculate all the pixel values of M^s in parallel. The graphics card also supports multiple texture units and rendering targets, which allows a shader to compute multiple silhouette textures at a time. Each silhouette texture computation uses two texture units on the graphics card and the ATI 4650 has a total of eight texture units available. The data sets used in this project contains images from four different cameras, meaning that it is possible to process all four views at the same time.

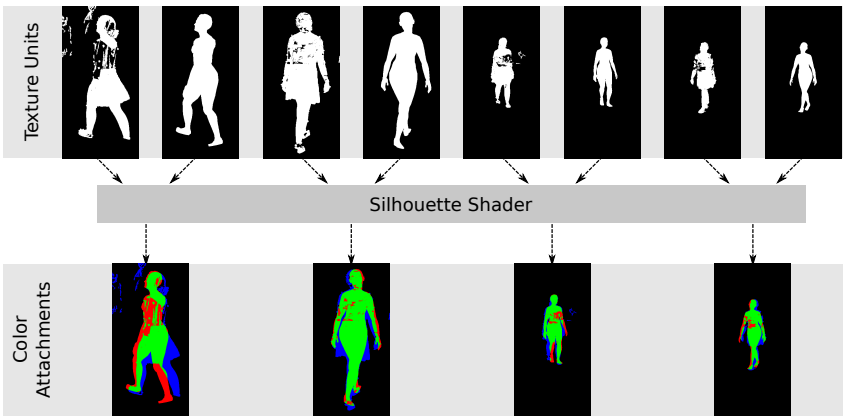


Figure B.3: *Silhouette Shader set up.*

To make use of these capabilities, each set of background subtracted images and pose guess

images for the four views are attached to the eight available texture units. The silhouette results are then written in each of the four color attachments of the framebuffer object. Each of the color attachments are textures, which can now be send to the summation function, described in section B.2.2, where the true values of R_t, B_t and Y_t will be calculated. The structure of the set up can be seen in figure B.3.

B.2.2 Average

In the following section an algorithm for summing and averaging over the `silhouette_texture`'s with OpenGL is proposed. The proposed algorithms will not return anything to the system memory due to the slow transfers between the video memory and system memory. Instead will the calculated probabilities be placed in another so called `probability_texture` ordered after the particle number.

To find the average likelihood, one has to find the sum of the non- and overlapping regions R_t, B_t and Y_t cf. B.1,B.2 and B.3. There is no easy way to sum up all the pixels in a texture in parallel, due to the sequential nature of a summation. In order to solve this problem to some extend, it has been chosen to sum groups of four pixels up at a time over several iterations until there is only one pixel left. This approach has the advantage to be parallel without having problems with concurrent write to the same pixel, which would lead to erroneous results.

An illustration of the proposed algorithm is found in figure B.4. It is easy to see that the above

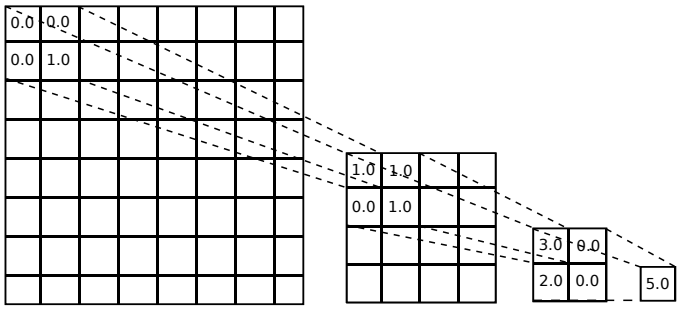


Figure B.4: *Proposed summation algorithm. A group of four pixels are summed in each iteration. In this case four iterations are needed to find the total sum*

summation algorithm has a drawback, i.e. it only works effectively on textures with the width and height equal to 2^k where k is an integer. If the texture is not a power of two, sooner or later the texture can not be divided by two resulting in uneven texture sizes. To deal with this, if a texture can not be divided by two, its width and height will be saved to two variables `real_width` and `real_height` and then added by one in both width and height which will result in a texture size dividable with two. The smaller texture will now have the width $(\texttt{real\_width}+1)/2$ and height $(\texttt{real\_height}+1)/2$. The shader rendering the small texture from the bigger texture, should use the variables `real_width` and `real_height` to avoid reading from a non existing pixel. This approach is illustrated in figure B.5.

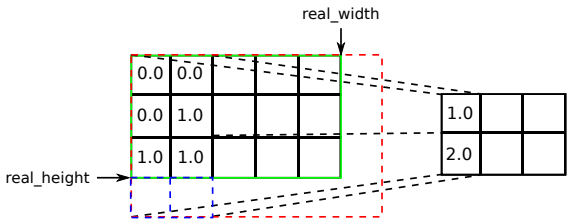


Figure B.5: *The summation shader rendering to the smaller texture to the right reading from the bigger texture, ignores the pixels located in a position greater than real_width and real_height (illustrated in the blue box). This cause the shader only to sum the pixels within the bigger texture even though it is not a power of two texture (illustrated as the green box)*

With the above approach, the shader is able to find the sum of a texture, even if it is not a power of two texture. For an initial texture of the size $1280x720$ (used in the HumanEva project), results in 11 iterations.

Recall that the silhouette textures were generated with each region in each color-channel, see B.2.1. This is not an issue for the summation algorithm given that OpenGL natively differentiates between the different color channels, resulting in a texture with the size $1x1$ with three channels. One channel for each region R_t, B_t and Y_t.

To speed up the summation algorithm up even further, the approach used in the silhouette algorithm B.2.1 using more than one texture unit on the GPU, is used. It is known that the likelihood has to be calculated for each view, which will naturally lead to the use of four texture units, i.e. one texture unit for each view. An illustration of the utilization of four texture units with the four views, calculating R_t, B_t and Y_t with only one shader call per iteration, is shown in figure B.6.

Note that in order for the summation shader to work properly, the textures have to be of the type `GL_RGBA32F_ARB` to avoid overflow of the pixels.

The four outputs from the summation algorithm are just a number representing the sizes of the different regions R_t, B_t and Y_t. To find the average likelihood for the four views, one last shader is needed. This shader has to input the four textures with the size $1x1$ and calculate the average likelihood as follows:

$$-logp(y_t|x_t)^v \propto (1-\alpha)\frac{B_t}{B_t+Y_t}+\alpha\frac{R_t}{R_t+Y_t}$$
$$-logp(y_t|x_t) = \frac{1}{4}\sum_v -logp(y_t|x_t)^v$$

Not only should the shader average over the four input textures and calculate the likelihood, but also place the resulting likelihood in a `probability_texture` depending on the particle currently propagating. Recall that the each texture from the summation shader contains all the sizes of each region in the RGB color channels. This enables the next shader only to depend on four three channel textures instead of 12 one channel textures.

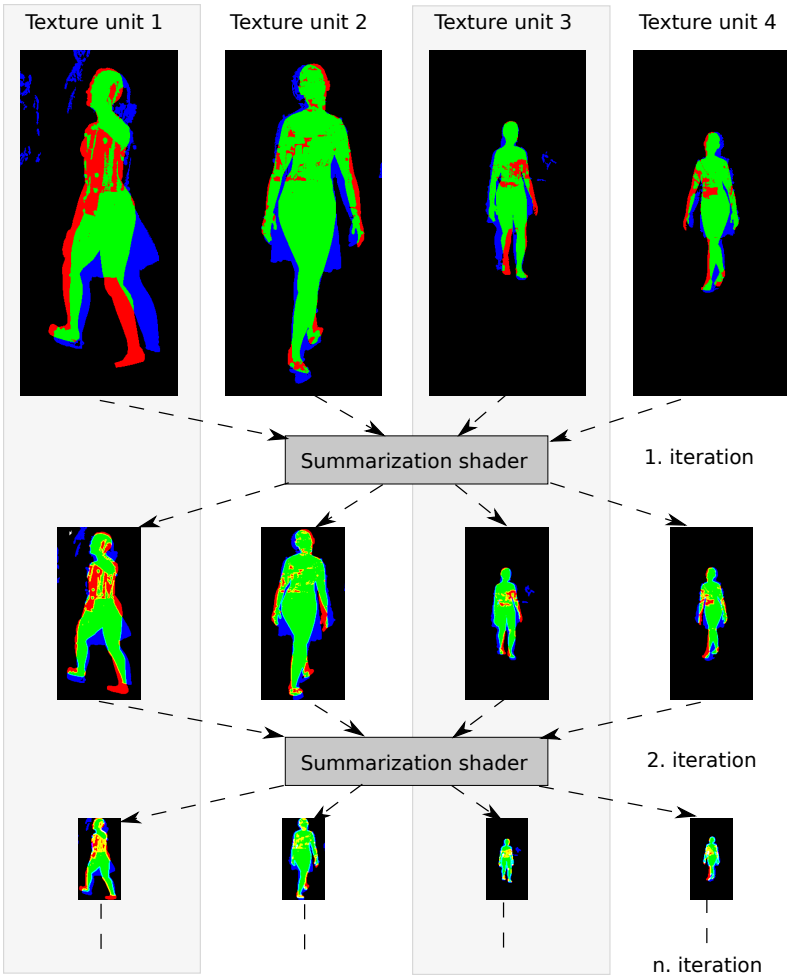


Figure B.6: *For each iteration, the summation shader adds four pixels and save the output in a texture half the size.*

The average shader is illustrated in figure B.7. After all the particles have propagated and the `probability_texture` has been filled with likelihoods, the function `gl_get_texture`, see section B.4.2, can be used to read the likelihoods into a standard Matlab array. Reading all the likelihoods from a texture instead of each individual likelihood, decrease the amount of slow video to system memory transfers.

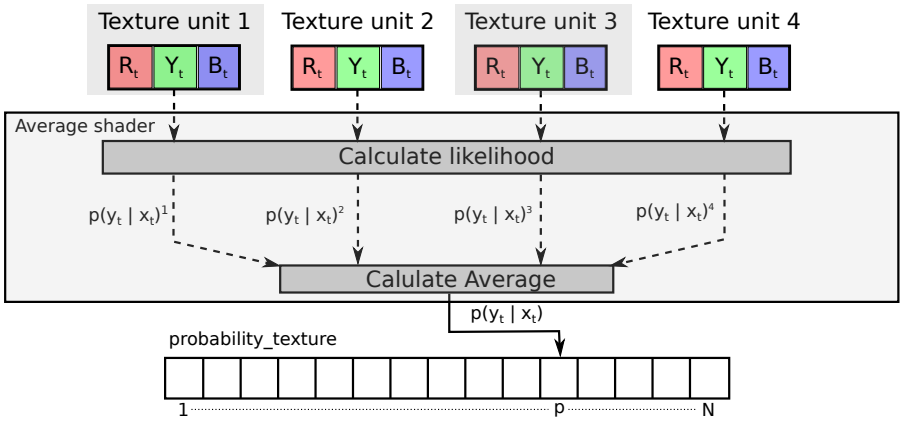


Figure B.7: *The likelihood of each four outputs from the summation shader is calculated and averaged. The average of the likelihood is saved into the probability_texture given the particle p. N being the total number of particles.*

B.3 c_skinning and c_projection

As can be seen in the timing discussion section A.2, the functions `m_skinning` and `m_projection` takes up a lot of computation time. While these functions would easily be implement in OpenGL, they have been tested to run very fast in a plain C implementation. Due to the fast C implementations, they will not be implemented in OpenGL. Both functions are relative simple so they will not be explained in details, but will be introduced in the following.

B.3.1 c_skinning

Each particle in the particle-filter is represented by a skeleton or pose. This skeleton needs to be skinned in order to use it in the likelihood function, see section B.2. The `skinning` function is used to skin the skeleton, where each skeleton represents a pose. The skinning is basically made out of various matrix and vector multiplications, where each vector or matrix represents the skin model, weights and skeleton model. The idea behind the matrix multiplication will not be explained in details, due to the goal of this project being optimizations.

The implementation of the various matrix multiplications in C using direct memory access, gave a huge boost in performance which will be evaluated in section C.

B.3.2 c_projection

The projection of the 3D mesh vertices from the `skinning` function into 2D image coordinates using various camera parameters, was done by using MATLAB's build in `project_points_2` function. As can be seen in section A.2, the `project_points_2` function is one of the slowest functions in the HumanEva project. In order to speed up the function to do the projections of the 3D vertices to 2D vertices, OpenCV's `ProjectPoints2` has been used instead.

To use an OpenCV function within a mex function environment, MATLAB's vertices and ma-

trices (`maxArray`) have to be converted to OpenCV's data container for matrices and vectors (`cvMat`). This has been accomplished without any data copying or conversion by sharing the data pointers between the cvMat structure and mxArray structure. Avoiding any data copying and conversion should help decreasing the overhead calling an OpenCV function from within MATLAB.

The new projection mex function is called `c_projection` and will, as explained above, interface with OpenCV's `ProjectPoints2` function. Using the `c_projection` function instead of MATLAB's build in function `project_points_2`, gave a huge speed up which will be evaluated in section C.

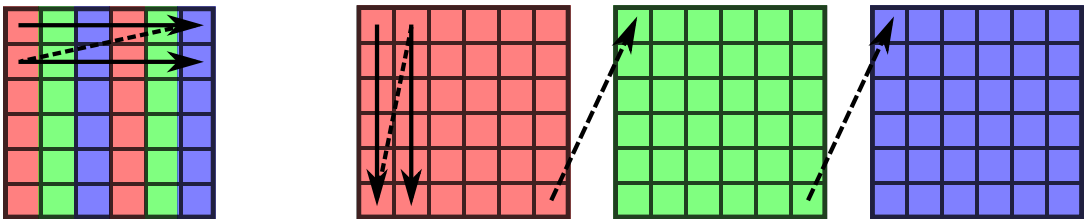
B.4 Misc OpenGL functions

B.4.1 gl_create_texture

To utilize OpenGL and its graphics capabilities we need to create textures in the graphics cards memory to allow computations on the GPU. To create these textures the function `gl_create_texture` is implemented as follows.

```
texture = gl_create_texture(gl_context,width,height,channels,[image])
```

The function returns an OpenGL texture ID in `texture`, referring to a texture on the graphics card of size `width` times `height` times `channels`.
If the optional argument `image` is given as input, `texture` will be populated with the data contained in `image` array else `texture` will contain zeros. `image` must be of size `width` times `height` times `channels`.
The function is a MEX function that interfaces with MATLAB and `image` is a MATLAB image array. This means that `image` is arranged as columns first in memory, arranging the image in memory as column by column for the width of the image, one color channel at a time, as in figure B.8(b). OpenGL reads image data row by row, reading all `channels` for each row position, as illustrated in figure B.8(a). To compensate for this difference the array `image` must be permuted with `permute(image,[3,2,1])`, before calling `gl_create_texture`, to ensure compatibility with OpenGL's methods for reading images from memory.



(a) *OpenGL image read order.*

(b) *MATLAB image read order.*

Figure B.8: *Visualization of how MATLAB and OpenGL expects to read images from memory.*

All textures created using this function are stored as RGBA floating point images in the graphics memory, as this format maps easily to the probability calculations in section B.2. The format is also native on newer graphics cards and allow us to store values larger than the range of the drawing functions, which we exploit in section B.2.2. The data provided in `image` must be unsigned byte, as this is the format expected by the `gl_create_texture`.

B.4.2 gl_get_texture

The function `gl_get_texture` has been created, in order to read an OpenGL texture back into a MATLAB array. The function is mainly used to download the `probability_texture` back to the system memory, which then can be used in the further computations. The function can be used as follows.

```
M = gl_get_texture(gl_context,texture)
```

Where `M` is a `[m x n]` matrix with m equals the height times the number of channels and n equals the width of the texture. For the `probability_texture` m would be number of particles and n would be one.

B.4.3 gl_init and gl_quit

Two very important functions are `gl_init` and `gl_quit`, whereas `gl_init` handles all the GLSL shader compilations, texture allocations and creating a `gl_context` struct, which is used throughout every OpenGL function calls in this project. `gl_quit` is used to deallocate all the textures on the GPU and safely close down the OpenGL context.

`gl_init` and `gl_quit` can be used respectively as follows.

```
gl_context = gl_init(width, height, show_window)
```

Where width and height would be the size of the pictures being computed and the `show_window` variable is a boolean controlling if a window with the framebuffer should be shown.

```
gl_quit(gl_context)
```

Appendix C

Timing evaluation after optimization

In order to evaluate the gain by porting parts of the MATLAB functions to plain C and OpenGL, timings have been done with the tic toc approach as described in section A.2. The computation time of the different functions can be seen in table C.1 and C.2.

Layer	#5	#4	#3	#2	#1
`m_image_initialization` + `gl_create_texture` [sec]	0.55	0.00	0.00	0.00	0.00
`c_skinning` [sec]	0.10	0.10	0.10	0.10	0.11
`c_2D_projection` [sec]	0.37	0.19	0.20	0.20	0.20
`gl_mesh_rendering` [sec]	0.98	0.49	0.96	0.40	0.45
`gl_likelihood` [sec]	2.68	3.11	2.64	3.20	3.16
Misc matlab code [sec]	2.36	1.44	1.14	1.03	1.00
Total pr. layer [sec]	7.04	5.33	5.04	4.93	4.92
Total pr. pose [sec]					27.26

Table C.1: Timing results for the Matlab functions over each layer for the first pose. For each layer the functions process 200 particles for each of the four camera views.

Layer	#5	#4	#3	#2	#1
`m_image_initialization` + `gl_create_texture` [sec]	0.53	0.00	0.00	0.00	0.00
`c_skinning` [sec]	0.10	0.10	0.10	0.10	0.10
`c_2D_projection` [sec]	0.20	0.19	0.19	0.20	0.20
`gl_mesh_rendering` [sec]	0.82	0.40	0.45	0.44	0.41
`gl_likelihood` [sec]	2.87	3.27	3.21	3.21	3.25
Misc matlab code [sec]	2.66	1.50	1.17	1.04	1.01
Total pr. layer [sec]	7.18	5.46	5.12	4.99	4.97
Total pr. pose [sec]					27.72

Table C.2: Timing results for the MATLAB functions over each layer for the second pose. For each layer the functions process 200 particles for each of the four camera views.

These results are compared to the old MATLAB functions in figure C.2.

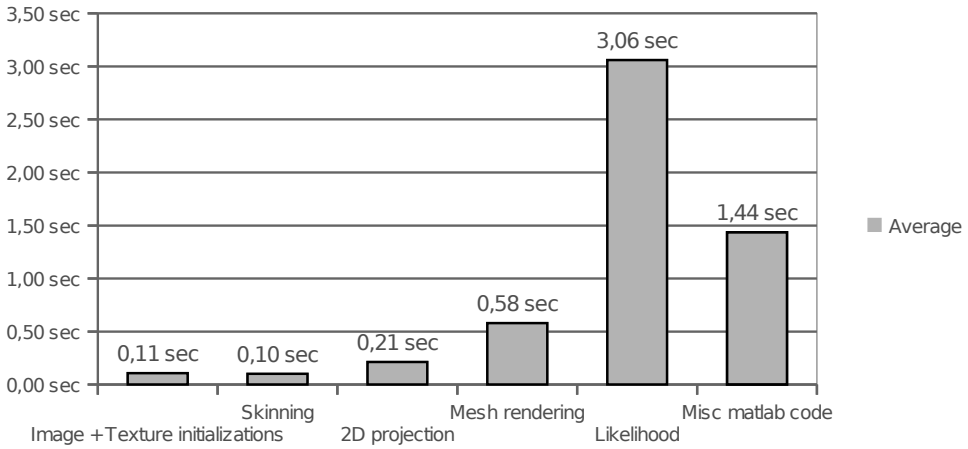


Figure C.1: *Average performance for the C and OpenGL functions computing two poses with five layers in each.*

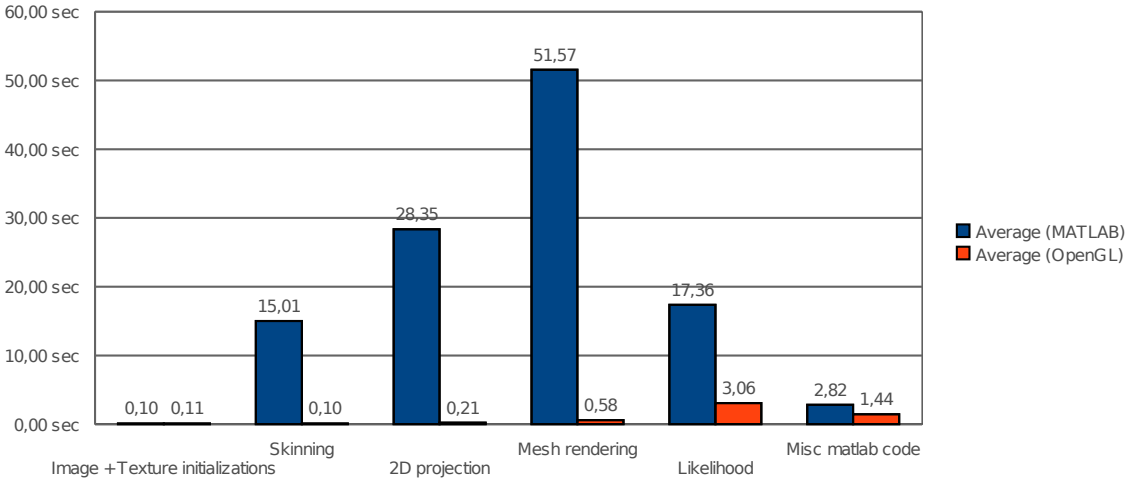


Figure C.2: *Comparement of the MATLAB functions versus the functions implemented in C and OpenGL, computing two poses with five layers in each.*

By comparing the timings from the pure MATLAB code and the timings obtained after all the performance optimizations it is clear that significant speed improvements have been made. By calculating the timing mean of the two pose calculations timed before and after the optimizations, meaning over the calculations of 2000 particles, an estimate of the performance gain can be made.

$$\bar{t}_{before} = \frac{572+580}{2} = 576 \tag{C.1}$$

$$\bar{t}_{after} = \frac{27.26+27.72}{2} = 27.49 \tag{C.2}$$

$$\Delta t = \bar{t}_{before} - \bar{t}_{after} = 548.51 \tag{C.3}$$

$$\bar{t}_{before} : \bar{t}_{after} = \frac{\bar{t}_{before}}{\bar{t}_{after}} \approx 21 \tag{C.4}$$

With a speed up of approximately 21 times, the performance gain is noticeable, but further investigation of tables A.1, A.2 and C.1, C.2 shows that the speed ups are not equally distributed over the reimplemented functions. Image and texture initializations and miscellaneous MATLAB code are areas where improvements have not been implemented and are therefore not of interest. A quick review of the remaining functions show that the functions for Skinning, 2D projection and Mesh rendering perform between 80 and 100 times faster than their equivalent MATLAB implementations, the Likelihood function only perform approximately 6 times faster than its MATLAB equivalent. The reason for the meager improvement in this function is caused by the internal `Average` function, see section B.2.2, which needs to run for 11 iterations to calculate the sum for images of the size $1280x720$.

Little can be done to improve the performance of the `Average` function. If the summation where to be done on the CPU instead, the `silhouette_texture`'s need to be transfered to the CPU. The `silhouette_texture`'s are high resolution pictures and for each layer 800 textures must be transfered, 200 particles in 4 views each. Using this approach a different bottleneck will become apparent. The problem is that the bus between the CPU and the GPU is designed for high upload speeds from the CPU, since usually the CPU will request large amounts of data to be computed on the GPU and then displayed on the screen, the CPU on the other hand rarely need any significant amount of data from the GPU and therefore has very limited bandwidth on the download direction. While most operations on the GPU can actually be processed while code is executed on the CPU, transferring data from the GPU to the CPU forces the GPU to finish all queued operations before it can return the requested data. In the meantime the CPU has to wait for the GPU to finish and return the requested data. The sheer amount of data to be transfered and the many state changes in OpenGL will make this approach less favorable, as the drawbacks will outweigh the gain from summing the values on the CPU.

Bibliography

[1] Smithsonian Institution. National air and space museum. http://www.nasm.si.edu/exhibitions/.

[2] Exparimentarium. http://www.experimentarium.dk/forsiden/udstillinger/.

[3] ZSL London Zoo. i-track. http://www.zsl.org/zsl-london-zoo/news/zsl-london-zoo-launches-i-track,432,NS.html.

[4] Duncan. Augmented reality at wellington zoo, 2007. http://theinspirationroom.com/daily/2007/augmented-reality-at-wellington-zoo/.

[5] The Dominion. Zoo ad helps hit lab win $2.7m deal, 2008. http://www.stuff.co.nz/technology/574577.

[6] ARToolworks. Artoolkit. http://www.artoolworks.com/.

[7] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[8] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc J. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[9] Paul A. Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR (1)*, pages 511–518. IEEE Computer Society, 2001.

[10] Liyuan Li, Kah Eng Hoe, Xinguo Yu, Li Dong, and Xinqi Chu. Human upper body pose recognition using adaboost template for natural human robot interaction. In *Computer and Robot Vision (CRV), 2010 Canadian Conference on*, pages 370 –377, 31 2010-june 2 2010.

[11] P. Cerri, L. Gatti, L. Mazzei, F. Pigoni, and Ho Gi Jung. Day and night pedestrian detection using cascade adaboost system. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1843 –1848, sept. 2010.

[12] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR (1)* [53], pages 886–893.

[13] Jianpeng Zhou and Jack Hoang. Real time robust human detection and tracking system. *Computer Vision and Pattern Recognition Workshop*, 0:149, 2005.

[14] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR (2)*, pages 1491–1498. IEEE Computer Society, 2006.

[15] Pedro F. Felzenszwalb, Ross B. Girshick, and David A. McAllester. Cascade object detection with deformable part models. In *CVPR* [54], pages 2241–2248.

[16] Bastian Leibe, Edgar Seemann, and Bernt Schiele. Pedestrian detection in crowded scenes. In *CVPR (1)* [53], pages 878–885.

[17] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *CVPR* [55], pages 1014–1021.

[18] Ronald Poppe. A survey on vision-based human action recognition. *Image Vision Comput.*, 28(6):976–990, 2010.

[19] Aaron F. Bobick and James W. Davis. The recognition of human movement using temporal templates. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(3):257–267, 2001.

[20] Alexei A. Efros, Alexander C. Berg, Greg Mori, and Jitendra Malik. Recognizing action at a distance. In *ICCV* [56], pages 726–733.

[21] Christian Thurau and Václav Hlavác. Pose primitive based human action recognition in videos or still images. In *CVPR* [57].

[22] Wei-Lwun Lu and James J. Little. Simultaneous tracking and action recognition using the pca-hog descriptor. In *CRV*, page 6. IEEE Computer Society, 2006.

[23] Nazli Ikizler, Ramazan Gokberk Cinbis, and Pinar Duygulu. Human action recognition with line and flow histograms. In *ICPR*, pages 1–4. IEEE, 2008.

[24] Hao Jiang and David R. Martin. Finding actions using shape flows. In Forsyth et al. [58], pages 278–292.

[25] Pingkun Yan, Saad M. Khan, and Mubarak Shah. Learning 4d action feature models for arbitrary view action recognition. In *CVPR* [57].

[26] Ivan Laptev and Tony Lindeberg. Space-time interest points. In *ICCV* [56], pages 432–439.

[27] Geert Willems, Tinne Tuytelaars, and Luc J. Van Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. In Forsyth et al. [58], pages 650–663.

[28] Matteo Bregonzio, Shaogang Gong, and Tao Xiang. Recognising action as clouds of space-time interest points. In *CVPR* [55], pages 1948–1955.

[29] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In Rainer Lienhart, Anand R. Prasad, Alan Hanjalic, Sunghyun Choi, Brian P. Bailey, and Nicu Sebe, editors, *ACM Multimedia*, pages 357–360. ACM, 2007.

[30] Heng Wang, Muhammad Muneeb Ullah, Alexander Kläser, Ivan Laptev, and Cordelia Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC*. British Machine Vision Association, 2009.

[31] Hueihan Jhuang, Thomas Serre, Lior Wolf, and Tomaso Poggio. A biologically inspired system for action recognition. In *ICCV*, pages 1–8. IEEE, 2007.

[32] Dave Wilson. *YUV Formats*. FOURCC, http://www.fourcc.org/yuv.php#YUY2.

[33] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. *Universal Serial Bus Specification*, 2000. sources/usb_20.pdf.

[34] Mian Zhou and Hong Wei. Face verification using gaborwavelets and adaboost. In *ICPR (1)*, pages 404–407. IEEE Computer Society, 2006.

[35] Michael J. Jones and Paul Viola. Face recognition using boosted local features, 2003.

[36] Chi-Chen Raxle Wang and Jenn-Jier James Lien. Adaboost learning for human detection based on histograms of oriented gradients. In Yasushi Yagi, Sing Bing Kang, In-So Kweon, and Hongbin Zha, editors, *ACCV (1)*, volume 4843 of *Lecture Notes in Computer Science*, pages 885–895. Springer, 2007.

[37] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In Ales Leonardis, Horst Bischof, and Axel Pinz, editors, *ECCV (2)*, volume 3952 of *Lecture Notes in Computer Science*, pages 428–441. Springer, 2006.

[38] Flowlib. http://gpu4vision.icg.tugraz.at/index.php?content=subsites/flowlib/flowlib.php.

[39] Manuel Werlberger, Thomas Pock, and Horst Bischof. Motion estimation with non-local total variation regularization. In *CVPR* [54], pages 2464–2471.

[40] Opengl 4.1 reference pages. http://www.opengl.org/sdk/docs/man4/.

[41] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artif. Intell.*, 17(1-3):185–203, 1981.

[42] Inria object detection and localization toolkit. http://pascal.inrialpes.fr/soft/olt/.

[43] Christoph H. Lampert. *Kernel Methods in Computer Vision*. now Publishers Inc., 2008.

[44] Lightsvm. http://svmlight.joachims.org/.

[45] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *ICPR (3)*, pages 32–36, 2004.

[46] Pittsburgh zoo & ppg aquarium. http://www.pittsburghzoo.org/.

[47] Ivan Laptev and Tony Lindeberg. Local descriptors for spatio-temporal recognition. In W. James MacLean, editor, *SCVMA*, volume 3667 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 2004.

[48] Leonid Sigal, Alexandru O. Balan, and Michael J. Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International Journal of Computer Vision*, 87(1-2):4–27, 2010.

[49] The MathWorks Inc. Introduction MEX-files. `http://www.mathworks.com/support/tech-notes/1600/1605.html#intro`.

[50] Opengl shading language. `http://www.opengl.org/documentation/glsl/`.

[51] The MathWorks Inc. tic, toc - measure performance using stopwatch timer. `http://www.mathworks.com/help/techdoc/ref/tic.html`.

[52] Song Ho Ahn. OpenGL Vertex Array. `http://www.songho.ca/opengl/gl_vertexarray.html`.

[53] *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*. IEEE Computer Society, 2005.

[54] *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. IEEE, 2010.

[55] *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE, 2009.

[56] *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*. IEEE Computer Society, 2003.

[57] *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE Computer Society, 2008.

[58] David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, editors. *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part II*, volume 5303 of *Lecture Notes in Computer Science*. Springer, 2008.