UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Computer Science Curriculum

Pihel Saatmann

Hand-tracking in Video Conversations

Bachelor's Thesis (6 ECTP)

Supervisor: Päivi Kristiina Jokinen

TARTU 2014

Hand-tracking in Video Conversations

Abstract:

This thesis describes some of the more common techniques for object tracking and an implementation of the colour-based tracking algorithm known as CAMShift. The algorithm is implemented as part of a simple object tracking plug-in for the video annotation tool ANVIL. The tracker can be used to automatically annotate hand gestures or the movements of any object that is distinguishable from its background. The plug-in records velocity, duration and total travel distance of hand gestures and outputs the recorded data to an annotation file. The tracker was tested on real recordings of dialogues and the results were compared to manually created annotations for hand gestures. Testing and evaluation revealed that data recorded by the tracker is not accurate enough to provide a complete alternative to manual annotation, but could rather be used as a basis for determining where hand gestures can be detected. Thus using the tracker in combination with a human annotator could significantly speed up the annotation process.

Keywords:

Object tracking, hand tracking, CAMShift algorithm, gesture annotation, gesture detection, ANVIL, plug-in,

Käeliigutuste tuvastamine ja jälgimine videosalvestistes

Lühikokkuvõte:

Antud bakalaureusetöö kirjeldab mõningaid meetodeid objektide jälgimiseks videosalvestistes ning värvipõhise jälgimisalgoritmi CAMShift realisatsiooni. Algoritm on realiseeritud lisamoodulina annoteerimisvahendile ANVIL. Loodud tööriist suudab jälgida käsi ning muid värvilisi objekte ning automaatselt tuvastada liigutusi videosalvestistes. Liigutused annoteeritakse automaatselt ning info liigutuse alg- ning lõpppunkti, keskmise kiiruse ning läbitud teekonna kohta kirjutatakse ANVIL'i annotatsioonifaili. Tööriista testiti videosalvestiste peal kahe inimese vahelisest suhtlusest ning tulemusi võrreldi käsitsi tehtud annotatsioonidega. Tulemuste hindamisel selgus, et tööriistaga tehtud annotatsioonid ei ole piisavalt täpsed, et neid saaks kasutada käsitsi tehtud annotatsioonide asemel. Tulemused on siiski piisavalt täpsed, et neid kasutada baasina käeliigutuste tuvastamiseks. Seega on valminud tööriist mõeldud pigem manuaalse annoteerimisprotsessi lihtsustamiseks ning kiirendamiseks.

Võtmesõnad:

Objektide jälgimine, käeliigutuste jälgimine, CAMShift algoritm, käeliigutuste annoteerimine, käeliigutuste tuvastamine, ANVIL, lisamoodul

Table of Contents

Introduction....4

1 Object Tracking....5

1.1 Object Detection....5

1.2 Tracking Methods....6

1.3 The Viola-Jones Framework....7

2 CAMShift Algorithm....9

2.1 The Mean-Shift Algorithm....10

2.2 CAMShift for Tracking coloured Objects....11

2.3 Solutions to Object Tracking Problems....12

2.4 Limitations....13

3 Implementation....14

3.1 Technologies Used....14

3.2 Plug-in Description....15

3.2.1 Creating the Template Histogram....15

3.2.2 Image Processing and Tracking....16

3.2.3 Detecting Movement....17

3.3 Plug-in User Interface....18

4 Evaluation....21

4.1 Tracking Bare Skin and coloured Objects....21

4.2 Tracking With Different Video Encodings....22

4.3 Comparison to Manual Movement Detection 23

5 Discussion and Future Work....25

Summary....27

References....29

Appendix 1 – Starting the Plug-in....31

Appendix 2 – Additional files....32

Introduction

The goal of this thesis is to describe various methods for tracking objects in video sequences and to implement one of the methods for automatic gesture annotation. The implementation has been made as a plug-in for the ANVIL annotation tool [1] and can be used for recognising and annotating hand gestures in recorded video conversations. Hand gesture recognition consists of several elements such as hand detection, movement tracking and gesture detection. The implementation can track and detect gestures, however initial detection of the hand inside the video frame is left to the user. The plug-in has been tested on real dialogue data recorded as part of the Estonian Science Foundation's project MINT (Multimodal INTeraction, ETF8958). The aim of the MINT project is to study signals of multimodal communication (hand gestures, head movements etc) and their relation to speech [2]. Annotation of video data is an important prerequisite for human communication studies, but doing this manually is time and resource consuming. It is thus important to study automatic tools for annotation.

Section 1 gives a general overview of object detection and tracking and describes an object detection method known as the Viola-Jones framework. Section 2 describes in detail the CAMShift algorithm which was selected for the implementation. Section 3 gives an overview of the plug-in implementation and Section 4 discusses evaluation on the project MINT data. Section 5 outlines the problems encountered during testing the plug-in, possible solutions to said problems and future improvements to the implementation. The source code for the implementation and necessary installation files are included as an archive.

1 Object Tracking

Object tracking is a sub-domain of computer vision that deals with locating the position of an object in each consecutive frame of a video. A tracking algorithm usually consists of detecting the object, tracking it and estimating it's movement trajectory. Additionally information about the object's properties such as orientation and shape can also be provided.[3]

Efficient algorithms have to overcome problems such as image noise, poor or changing lighting, complex object shapes, irregular object motion, occlusion and distractors present in the video and be able to run in real-time. Tracking may be simplified by setting constraints on the motion or appearance of the object (for example only including objects of a certain colour or shape or assuming the motion is smooth).[4]

1.1 Object Detection

An object can be represented by its shape and appearance. Some examples of shape representation include a set points or a single point, primitive geometric shapes and contours that mark the edge separating the object from the rest of the image. Appearance can be represented by probability densities, appearance models or templates that are formed using shapes of silhouettes. The advantage of templates and models is that they also contain information about the shape. Representations are usually chosen according to the object domain.[3]

A relevant piece of information (shape or appearance representation) about an object is called a feature. In order for an object detector to be successful a suitable feature or features need to be selected for analysis. The best kind of visual features are ones that enable objects to be more easily distinguished. Features are mostly selected manually depending on the object domain, but automatic feature selection by using supervised learning is also possible. An example of using supervised learning is the Viola-Jones object detection framework that is described in section 2.3.[3]

Object detection is usually based on information from a single frame. In order to reduce the number of false detections, image differencing information which highlights changing regions in a sequence of frames can also be used. The method of detection depends on the set of used features. An example of a detection method is image segmentation which partitions the image into perceptually similar regions in order to find object contours.[3]

1.2 Tracking Methods

Detecting an object and matching its instances across frames can be done separately or jointly. In the first case, detection methods are applied to find possible objects in each frame and then a tracking method tries to match the instances across frames. In the latter case, the location estimation and instance matching are done jointly by iteratively updating the object's location and region information based on previous information. Both of these cases use shape and appearance models to represent objects. [3]

Tracking methods can be divided into three categories – point, kernel and silhouette tracking. A detailed taxonomy of tracking methods can be seen in Figure 1.[3]

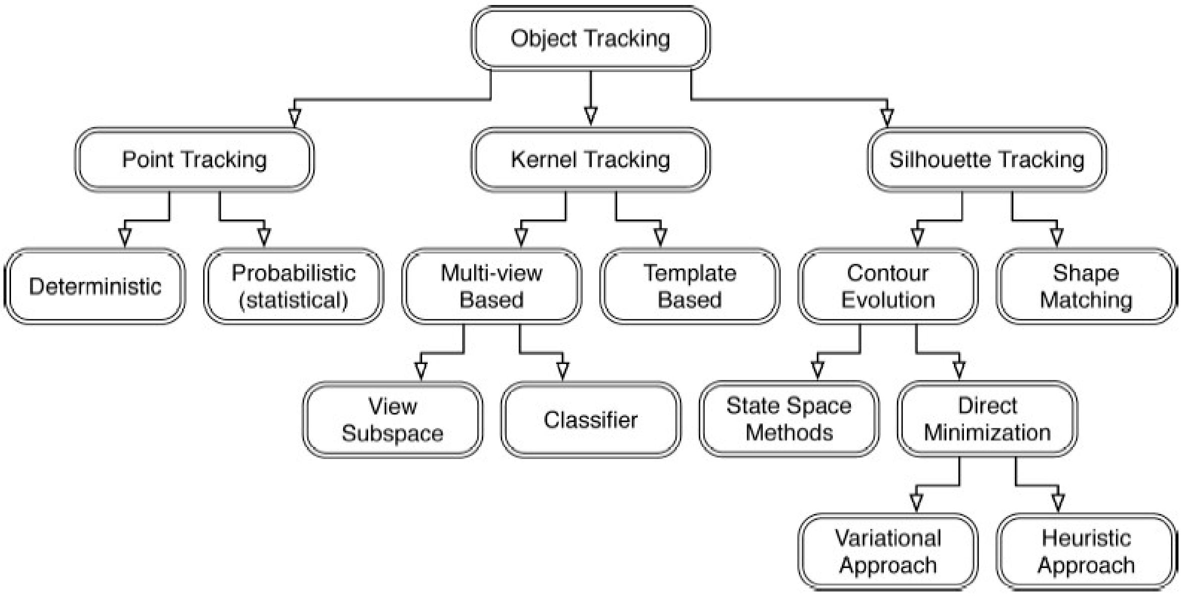


Figure 1 - Taxonomy of object tracking methods. Taken from [3]

Point tracking requires a detection method to find objects in each frame. Objects are represented by sets of points and the tracker attempts to match the features in consecutive frames based on the previous object state (position, motion direction etc).[4]

Kernel tracking is performed by computing the motion of an object's primitive shape or appearance features in consecutive frames. An example of kernel tracking is template matching which finds the region of interest in a video frame and searches the next frame for a match. The CAMShift algorithm described in section 3 uses kernel tracking.[4]

Silhouette tracking uses object models such as colour histograms or object contours that are calculated in each frame to the find the object region in the next frame. This method can be used for objects that are too complex to be represented by points or primitive shapes.[4]

1.3 The Viola-Jones Framework

The Viola-Jones framework is an example of a feature-based tracking method that was published in 2001 by Paul Viola and Michael J. Jones. It was primarily meant as a face detection method although it can also be used to detect other types of objects.[5]

The framework classifies objects based on rectangle features (Haar-like features), which are represented by the sums of pixel values in the rectangular areas. The value of each feature is the sum of pixel values within the white rectangles subtracted from the sum of pixel values in the dark rectangles. The framework uses an image representation called the *integral image.* At a given location the integral image represents the sum of all pixel values above and to the left of the given location. This allows for any feature at any scale and location be evaluated with a few operations. If the difference between the dark and light regions is above a predetermined threshold then a feature is detected. A feature used for face-detection could for example compare the intensity of the eye regions to the intensity across nose region in a frontal face image (represented by the rightmost column in Figure 2).[5]

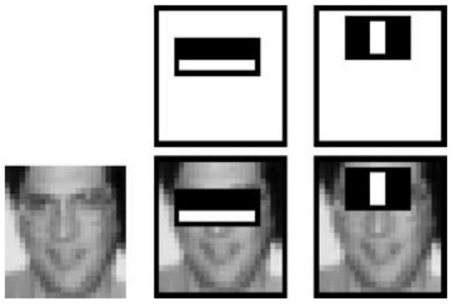


Figure 2 - Viola-Jones feature examples.[5]

If all possible combinations of position, scale and feature type in a particular sub-window of an image would be considered then the amount of features to be evaluated each time would be very large. Therefore in order to reduce computing time the framework uses a cascade of simpler (also called weaker) classifiers to form a strong classifier. The detection process works on the principle that if the first classifier gives a positive result then the next one is applied and so forth. A negative result will reject the sub-window. Features are chosen and classifiers are trained by a modified version of a machine learning algorithm called AdaBoost (short for “Adaptive Boosting”). Each next classifier in the cascade is trained using the examples which pass through the previous classifiers. In this way the majority of the image sub-windows would be rejected in the earlier stages so the more complex classifiers would not have to be applied.[5]

In case of object tracking the detector would be applied to each sequential video frame to attempt to detect and locate the object. The framework would have to be retrained on a selected data-set for it to work on different object types (for example a classifier to be trained on frontal face images would not work on profiled faces).

2 CAMShift Algorithm

The CAMShift (Continuously Adaptive Mean-Shift) algorithm was developed by computer vision researcher Gary R. Bradski in 1998. It was intended to be used as a face-tracking algorithm for a computer vision interface for controlling video games and exploring 3D virtual environments. In this chapter I will review the algorithm. The presentation is based on the references [6], [7] and [8].

Since the CAMShift tracker was intended to be used in real time and as part of a larger user interface, it needed to be fast and computationally efficient. Complex methods such as feature matching were not efficient enough nor necessary for the kind of basic object tracking the algorithm was meant for. Therefore a colour-based tracking method was chosen.

The CAMShift algorithm builds on a feature-space analysis method called mean-shifting (see section 2.1). Figure 3 shows how CAMShift is used for object tracking. For video sequences the algorithm is applied to each frame to find the new location of the tracked object.

Steps of applying CAMShift:

1. The image information is converted into a probability distribution.
2. An initial search window size and location for mean-shifting are chosen.
3. The mean-shift algorithm is applied to the distribution data.
4. The next window size is calculated based on the distribution inside the search window.
5. Steps 3 and 4 are repeated until the window doesn't move anymore or moves less than a predefined threshold.

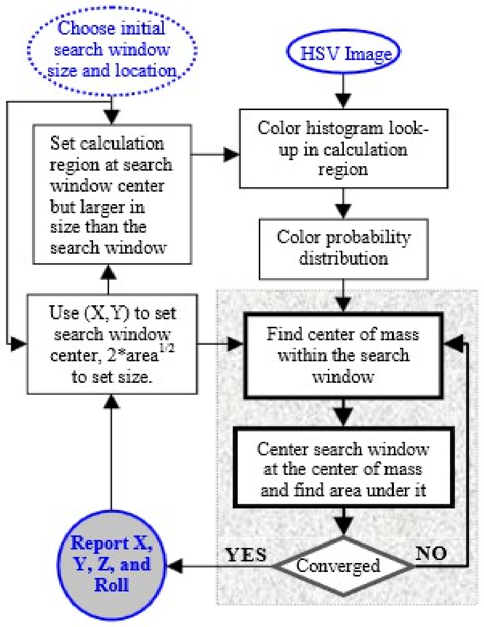

Choose initial search window size and location
HSV Image
Set calculation region at search window center but larger in size than the search window
Color histogram look-up in calculation region
Color probability distribution
Use (X,Y) to set search window center, 2*area^1/2 to set size.
Find center of mass within the search window
Center search window at the center of mass and find area under it
YES
Converged
NO
Report X, Y, Z, and Roll

Figure 3 - CAMShift for coloured object tracking.[6]

2.1 The Mean-Shift Algorithm

The mean-shift algorithm is a „robust non-parametric technique for climbing density gradients to find the mode (peak) of the probability distribution“ [6]. This means that if given a set of data-points the algorithm associates each point with a nearby maximum of the underlying probability density function and iterates a predefined search window over the data. After each iteration the search window moves to a more dense region until it is centred over the maximum. The mean-shift algorithm is represented by the grey box in Figure 3.

How mean-shift is calculated:

1. Search window size and location are selected.
2. The mean of the data inside the search window is calculated.
3. The window is centred at the location of the mean value.
4. Steps 3 and 4 are repeated until the window doesn't move anymore or moves less than a predefined threshold.

The robustness of the algorithm means that it ignores outliers in the data - in case of mean-shift data-points far away from the search window are ignored [8]. This helps eliminate distractors in a video sequence that are not in direct contact with the tracked object (for example faces in case of tracking a hand).

The mean-shift algorithm can be used for finding the location of the object in static images, however in a video recording objects can move away or towards the camera or rotate so they change in size. In those cases the size of the search window would also have to be changed and using mean-shift would fail. As a solution the CAMShift algorithm has added a dynamic search window that adjusts itself to changes in the tracked object’s size.

2.2 CAMShift for Tracking coloured Objects

CAMShift uses the HSV (Hue Saturation Value) colour system (Figure 4**)**, which separates the colour (hue) of a pixel, concentration of the colour (saturation) and brightness (value) into three channels. The values taken from the hue channel are stored in a 1D histogram, which is later used as a colour model to convert the hue values of pixels in a video frame to a colour probability distribution.

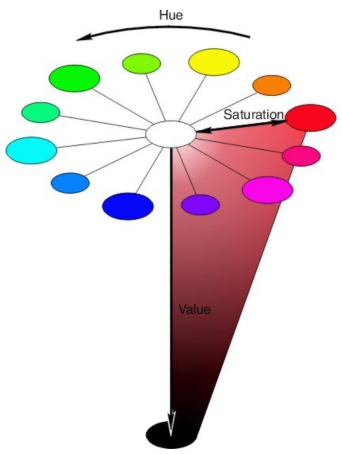


Figure 4 - HSV colour model.[9]

A histogram is a table of possible hue values. Each entry in the histogram represents how many pixels in an image have that specific hue value. If the histogram is normalised so that the sum of the entries equals 1 then each entry represents the percentage of pixels with the given value in an image. In other words a normalised histogram can be used to represent a probability distribution of the colour data.[8, p.90]

If we want to track an object, we create a histogram of all the pixels representing the given object and normalise it. Then for the next video frame the probability of each pixel belonging to the tracked object is calculated (the probability distribution of the image). For example if 40% of the pixels in the tracked object have a specific hue value then all the pixels in the next video frame with the same hue value have a 40% probability of belonging to the tracked object.

The probability distribution is then given to the CAMShift algorithm, which iterates the search window over the image and finds the location where the probability inside the window is maximised. To make tracking more efficient the colour probability distribution doesn't have to be calculated for the entire image, but can be restricted to a region surrounding the search window.

The quality of the tracking depends on the quality of the probability distribution. A problem with using the HSV colour system is that if the brightness value is very low then saturation is also low and hue cannot accurately represent the colour differences between pixels. This also applies to pixels that have very high brightness (white pixels) or just very low saturation (neutral grey pixels). These pixels contribute to noise and therefore have to be filtered out by applying saturation and brightness thresholds.

2.3 Solutions to Object Tracking Problems

The CAMShift algorithm provides solutions to some of the most common object tracking problems. Most image noise is filtered out by using the HSV colour model and by correctly applying thresholds. CAMShift has been shown to track the object's X and Y position and angle quite well in up to 30% of white noise [6]. Use of the HSV colour model and thresholds also gives the tracker a wide lightning tolerance.

The robustness of the algorithm allows it to ignore distractors that are outside of the search window. Distractors inside the search window, but not in direct contact with the tracked object tend to be ignored as well. Occlusion is handled by the algorithms tendency to move to the nearest dominant peak of the distribution. This means that if a distractor passes in front of the tracked object the tracker should stay on the tracked object as long as it's not completely occluded or the probability distribution of the distractor doesn’t provide a “better match”. In case of partial occlusion the tracker tends to “stick to the mode of the colour distribution that remains” [6].

CAMShift has the tendency to grow or shrink its search window to contain the connected probable pixels of a tracked area. This means that the search window doesn't stick to just a small part of the object, but grows to track the entire object (in case of tracking a hand if the search window is initially set on the fingers, it moves to encompass the entire hand). Thus if the initial selected part of the object is occluded, the algorithm is still able to track the remainder. This is also useful when tracking objects that change in size as their position or rotation changes during a video sequence.

2.4 Limitations

CAMShift tends to fail as a tracker if the video frame has a large amount of probable pixels (if the background has the same hue as the object or there many other objects with a similar hue in the image). Too bright, dim or coloured lighting and changes in lighting during the video can also cause tracking errors.

The search window's tendency to encompass the connected probable pixels doesn't allow for more specified tracking (for example tracking only part of the face) and can cause the tracker to move to an unwanted object or part of the background.

If the tracked object is lost (for example when it moves too far from its last detected location between two video frames) or the tracker moves to another object, some other method of object recognition has to be used or the search window needs to be manually relocated. It is sometimes possible that the object is automatically relocated when it moves back towards its last known location.

3 Implementation

CAMShift was chosen for the implementation, because the main purpose of the plug-in is to detect movement and knowing the precise hand position in the 3D-space is not required. colour-based tracking allows to estimate the approximate position of the object with enough precision to detect movement between frames. Unlike the Viola-Jones tracker CAMShift does not require prior training or a database of features to work and could also be used to track various other coloured objects besides hands.

Currently ANVIL has a face-tracking plug-in that creates automatic annotations for head movements. It uses the face recognition algorithms implemented in OpenCV that are an improved version of the Viola-Jones face detector. The tracker can also be used for other body parts, however the available settings do not include a feature for hands.[10]

3.1 Technologies Used

Java - the plug-in is uses Java version 7.

OpenCV (Open Source Computer Vision) is a C and C++ library of algorithms for computer vision, video analysis and image manipulation. It has the implementations of many object recognition and tracking algorithms (including CAMShift).[11] The plug-in uses version 2.4.3.

JavaCV provides wrappers to the OpenCV and other libraries used in computer vision. It wraps C API wherever possible, and C++ API when necessary. It also includes helper classes and methods on top of OpenCV to facilitate its integration to the Java platform.[12] The plug-in uses version 0.3.

ANVIL is a free video annotation tool that allows the user to create multi-layered colour-coded annotations. ANVIL also features several tools that allow the annotators to analyse and manage the annotations, for example calculate the intercoder agreement (kappa score) and association tables for annotation categories. The annotations can be written out in XML-format and so be easily used for further statistical analysis.[1] The plug-in has been tested with ANVIL version 5.1.7. ANVIL can be downloaded at http://www.anvil-software.org/.

3.2 Plug-in Description

The tracker's work can be divided into three segments – creating a histogram to be used as the object template, converting the video frames to an appropriate probability representation and applying CAMShift, detecting object movement and writing movement data to an output.

The OpenCV library contains an implementation of the CAMShift algorithm. Before an image can be used as input for the CAMShift algorithm it has to be converted into a suitable format that represents it's colour probability distribution (a probability map). OpenCV also provides functions for image processing and histogram creation.

OpenCV methods use a special image structure called the IplImage which has multiple formats for representing different pixel depths, number of colour channels and other image parameters [13]. For using CAMShift all video frames have to converted to the HSV colour space before information is extracted from them.

3.2.1 Creating the Template Histogram

Before tracking can start, the program needs to create a hue histogram of the tracked object to be used later as a template for colour probability calculations. This is done simply by having the user mark a rectangular region in the image that contains the object and then calculating the histogram for the marked region. The marked region can be smaller than the object as CAMShift will automatically expand the window to the largest possible coloured area that matches the colour information of the marked region. If the region is bigger than the tracked object then colour information from the background objects may impair tracking. In order to achieve maximum precision the user should try to mark a region that contains as much of the tracked object as possible and no background. If the tracked object is not rectangular in shape it is always best to select a smaller region so that not background areas are included.

After the tracked object has been marked the image has to be converted to the HSV colour space and split into three separate channels (hue, saturation and value). Values from the hue channel are then put into a one dimensional histogram. OpenCV uses hue values ranging from 0 to 180 so the resulting histogram has 180 bins (entries for different hue values). If a saturation threshold is set then pixels with low saturation are ignored by applying a saturation mask on the hue channel, which removes all pixels with a saturation value below the threshold. The histogram is then normalised so that the values range from 0 to 1 to make probability calculations simpler.[9]

3.2.2 Image Processing and Tracking

After the template histogram is created, the program will take the next frame from the video sequence and process it. For each frame the tracker has to:

1. Convert the image to HSV colour space.
2. Calculate the colour probability map for the image.
3. Eliminate pixels with low saturation from the map.
4. Apply CAMShift to the probability map.
5. See if the object has moved a distance that is bigger than the movement detection threshold.

After converting the image to HSV it is separated into three channels. The saturation channel is used to create a saturation mask for the image which is later used to eliminate pixels that have a value below the saturation threshold.

Next the probability map is created by applying a technique called histogram back-projection, which uses the template hue histogram generated at the beginning of tracking. This creates an image where regions with highest probability are marked with white colour and regions with lowest probability with black, greyish pixels have a probability somewhere between those. The saturation mask is applied to the back-projection image to filter out pixels with low saturation (Figure 5).[9]

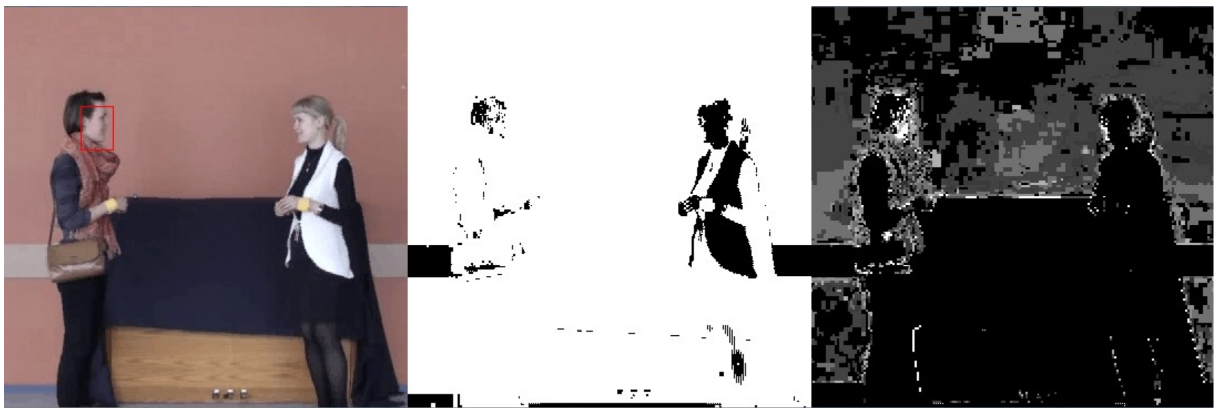


Figure 5 - normal view, saturation mask and back-projection for skin hue.

OpenCV's implementation of CAMShift takes the back-projection, location of the search window in the previous frame and search termination criteria as input and outputs the window's new location. It also returns the number of iterations it took for the tracker to find the window location. The implementation has two termination criteria – the maximum number of iterations and the window movement distance below which the window is considered to be centred on the new location.[13]

3.2.3 Detecting Movement

The search window's location is compared to its location in the previous frame. If the distance between both locations is above a predetermined threshold then movement is detected and the start of the movements is marked. In order to detect slow movements more accurately the next few frames that have a value below the threshold are still counted as part of the movement if their total move distance is above the threshold. The end of the movement is marked by the next frame that has a distance value below the threshold. Very small movements that last less than 2 frames or have a total distance below the movement threshold are ignored since they are most likely caused by bad video quality or changes in lighting.

The start and end positions and total movement distance and duration are saved and written to the selected annotation track as a new interval (see Figure 6). The start and end locations of the movement (2 points in the 2-dimensional space) are also displayed on the main video window during each interval. ANVIL enables the user to customise the appearance of the points displayed on the video (for example a vector between the start and end points).

3.3 Plug-in User Interface

Figure 6 shows the user interface for ANVIL and the controls for the hand-tracker. The user can modify the tracker's settings in order to increase its efficiency and accuracy.

1 Main ANVIL window.

2 Main video window.

3 Annotation tracks.

4 Annotation interval info, which displays movement data:

- **4.1** Distance in pixels.
- **4.2** Movement end and start points.
- **4.3** Average velocity of the movement (pixels/frame).
- **4.4** A vector connecting the start and end point of a movement.

Tracker controls

5 Minimum saturation threshold (see section 3.2.2).

6 The number of frames to skip on each iteration.

7 Movement threshold in pixels (see section 3.2.2).

8 Toggles the back-projection display (selected by default).

- **8.1** The back-projection display.

9 Toggles the saturation display.

10 Enables object tracking (selected by default).

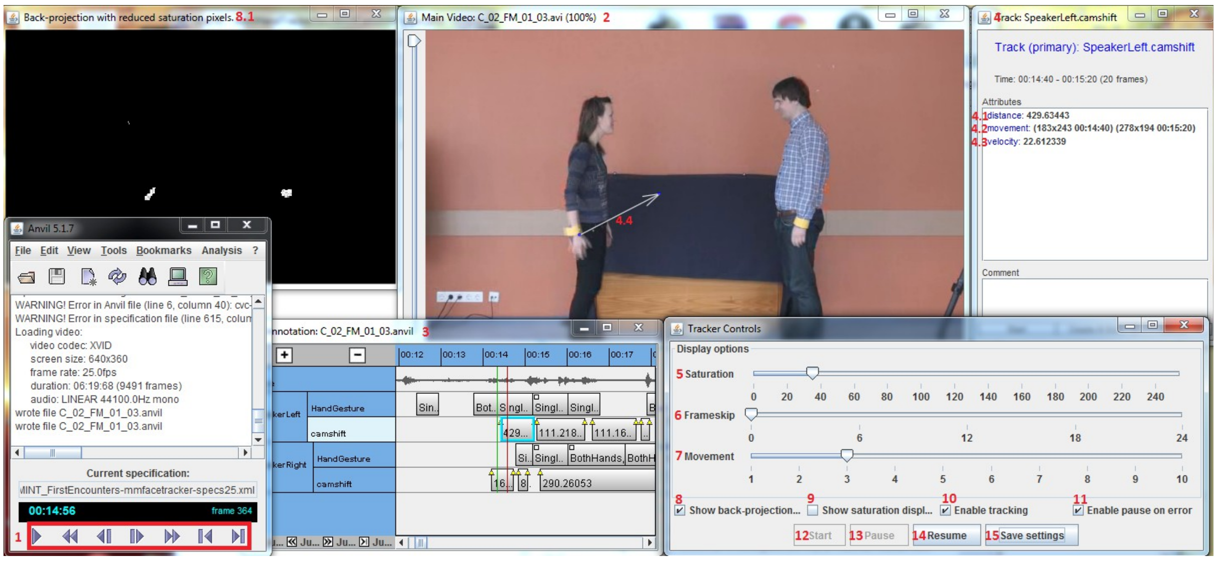
11 Enables pausing when search window is lost (selected by default).

12 Starts the tracking.

13 Pauses the tracking.

14 Resumes the tracking.

15 Saves tracker settings to the selected track.


Back-projection with reduced saturation pixels. 8.1
Main Video: C_02_FM_01_03.avi (100%) 2
4.4
4rack: SpeakerLeft.camshift
Anvil 5.1.7
File Edit View Tools Bookmarks Analysis ?
Annotation: C_02_FM_01_03.anvil 3
Tracker Controls
Display options
5 Saturation
6 Frameskip
7 Movement
8 Show back-projection...
9 Show saturation displ...
10 Enable tracking
11 Enable pause on error
12 Start
13 Pause
14 Resume
15 Save settings
1
HandGesture
camshift
00:14:56
frame 364

```
WARNING! Error in Anvil file (line 6, column 40): cvc-
WARNING! Error in specification file (line 615, colum
Loading video:
    video codec: XVID
    screen size: 640x360
    frame rate: 25.0fps
    duration: 06:19:68 (9491 frames)
    audio: LINEAR 44100.0Hz mono
wrote file C_02_FM_01_03.anvil
wrote file C_02_FM_01_03.anvil
```

Current specification: MINT_FirstEncounters-mmfacetracker-specs25.xml

Track (primary): SpeakerLeft.camshift

Time: 00:14:40 - 00:15:20 (20 frames)

Attributes
4.1 distance: 429.63443
4.2 movement: (183x243 00:14:40) (278x194 00:15:20)
4.3 velocity: 22.612339

Comment

Figure 6 - ANVIL interface and tracker controls

Tracking cannot start until a template histogram has been obtained so if the user hasn't marked a tracked object region in the video and clicks the 'Start' button an error message is displayed in the application's console window. The user is allowed to re-mark the tracked object region before pressing the Start button. If the 'Show back-projection display' checkbox is checked the back-projection display will show a probability map of the current video frame. If the search window is lost during tracking the user can manually re-select the object in the video (note that this does not create a new template histogram).

In order to select another object to track or overwrite the template histogram the user has to close the “Tracker controls” panel and start the plug-in again from the Tools menu. Data already written to the track will not be lost with closing the control panel.

The default value of the saturation threshold is set to 65. Higher values of the saturation threshold help to filter out noise, but may also cause some pixels in the tracked object to be ignored if they have low saturation. The saturation and back-projection displays are meant to help the user decide on an appropriate saturation threshold and also to see how the background or other moving objects may be interfering with the tracking.

The default number of frames to skip is 0 which means that the tracking algorithm will be applied to every frame in the video. Increasing this value will lower the computation cost, but also increase the probability that the search window is lost or jumps to another object if the tracked object moves too much between frames.

The user can select the option to pause the tracking when the search window is lost (checkbox “Pause on error”). It is useful to leave this checkbox selected since it will ensure that the tracked object is not lost for an extended period of time. However when the tracked object is fully occluded the tracker will pause on every frame when this option is selected.

The “Enable tracking” checkbox will enable and disable the tracker’s work, but will not pause or resume the video.

The user can change the settings and pause or resume tracking at any time while the tracker is working. Buttons in the plug-in’s control panel should be used for pausing and resuming the video rather than controls in ANVIL’s main window.

4 Evaluation

The tracker was tested on videos recorded during the MINT project, the recorded participants have agreed on the use of the videos for research purposes and being shown to third parties. In each video two people standing face to face are filmed from the side so that their full body apart from the feet is in the frame. The raw videos recorded for the project were converted to two different formats (Xvid and Cinepak) and resized to 640x320 pixels. Videos with Cinepak encoding are of a lower quality than Xvid so the tracking window tended to be lost more frequently and tracking was not as precise. The tracker was not evaluated by tracking bare hands. Previous testing showed that video quality and the presence of multiple distractors and occlusion in the videos were not suitable for tracking with the CAMShift algorithm as the results produced were very inaccurate (see section 4.1).

Detailed results of applying the tracker to a 40-second sequence of one of the test video files are provided in section 4.2. Comparison of the tracker's ability to detect movements and detection precision to manually detected movement data is provided in section 4.3.

4.1 Tracking Bare Skin and coloured Objects

In some videos the recorded people had yellow bracelets around their hands, the tracker was tested on both these bracelets and on bare hands. When tracking the yellow bracelets the tracker was working as expected without any errors. The tracking window didn't move to the wrong object was lost only if the bracelet was completely occluded. Movement was detected relatively accurately.

Tracking bare hands proved to be more problematic. The tracker would jump to other hands or faces in the frame if the regions overlapped (for example if the people shook hands) or get lost if the hand was completely occluded (for example if the person put their hands in their pockets. Since the recording was made from a side angle then the hand further away from the camera would often interfere with tracking the other hand. Movement was not accurately detected and often small movement was detected when there actually was none.

Figure 7 shows comparison of the two tracking conditions. On the left is the colour probability image based on the bracelet and on the right based on the bare skin area of the hand. Tracking the bracelets proved more accurate, because the bracelets have a hue that is easily distinguishable from the background. In most cases it was the only object in the video of that particular colour. Looking at the probability images also shows that edges belonging to the background have a somewhat similar hue to the skin hue. This caused the tracker to sometimes get stuck to a part of the background when the hand became occluded. Things were made even worse if the person's clothes also had a similar hue. In those cases the tracker usually moved to the body region. Since the two bracelets in the video very rarely came into direct contact, the search window usually stayed on the correct object.

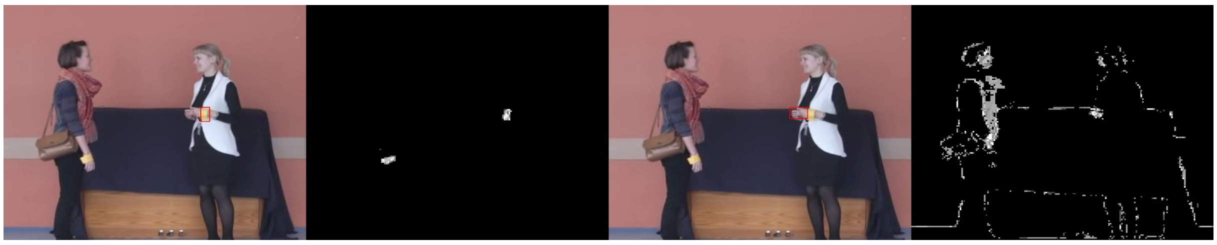


Figure 7 - Normal and probability map views for bracelet and bare skin tracking

4.2 Tracking With Different Video Encodings

Table 1 shows the result of testing the tracker on a video sequence that had the Xvid video encoding and a duration of 70 seconds (0:10 – 1:20) (file 'C_01_FF_01_02_Xvid.avi', see Appendix 2). The search window was set to the left person's bracelet. During the tracker's work the search window was never lost.

Table 1 - tracking with different movement thresholds with Xvid

Threshold	Movements detected	Average distance (px)	Average velocity (px/frame)
1	47	40	1,8
3	45	34	4
5	22	46	6

Table 2 shows the the results of testing the tracker on the Cinepak encoded version of the video (file 'C_01_FF_01_02_Cinepak.avi', see Appendix 2). With low thresholds the tracker often detected movement when there actually was none, which caused the algorithm to detect very long movements where there was actually none. The search window was lost multiple times during tracking (with each threshold) and had to be manually reset.

Table 2 - tracking with different movement thresholds with Cinepak

Threshold	Movements detected	Average movement distance (pixels)	Average movement velocity (pixels/frame)
1	21	517	6,8
3	124	84	7,9
5	131	47	8,2

4.3 Comparison to Manual Movement Detection

The tracker was applied to two different video files: C_01_FF_01_02 with duration 5:50 minutes and C_02_FM_01_03 with duration 6:17 minutes that had been previously manually annotated for hand gestures. The tracker was set to track the participants' yellow bracelet in both videos. In cases where one manually annotated movement was detected as multiple movements by the tracker it was counted as a true positive detection. Below is a comparison of the tracker's movement detection ability to the manual annotations. True positive values are marked as TP, false positive as FP and false negative as FN. True negative values were not counted as it would not be possible to count all the true negative detection instances in case of a video sequence.

Table 3 – 2-by-2 contingency table for left speaker in video C_01_FF_01_02_Xvid.avi

	Correct	**Not correct**
Detected	22 (TP)	68 (FP)
Not detected	0 (FN)	- (TN)

Table 4 – 2-by-2 contingency table for right speaker in video C_01_FF_01_02_Xvid.avi

	Correct	**Not correct**
Detected	9 (TP)	43 (FP)
Not detected	0 (FN)	- (TN)

Table 5 – 2-by-2 contingency table for left speaker in video C_02_FM_01_03_Xvid.avi

	Correct	**Not correct**
Detected	17 (TP)	24 (FP)
Not detected	0 (FN)	- (TN)

Annotation data for all three speakers was used to calculate the precision, recall and F-measure values for the tracker.

Precision or positive predictive value represents the fraction of correct detections over all movements that were detected:

$$P = TP \div (TP + FP) = (22 + 9 + 17) \div (22 + 9 + 17 + 68 + 43 + 24) \approx 0{,}262$$

Recall represents the fraction of correct detections over the number of movements that should have been detected:

$$R = TP \div (TP + FN) = (22 + 9 + 17) \div (22 + 9 + 17 + 0 + 0 + 0) = 1$$

Using precision and recall we can calculate the F-score to evaluate the tracker's accuracy on a scale of 0 to 1:

$$F_1 = 2 \times P \times R \div (P + R) = 2 \times 0{,}262 \times 1 \div (0{,}262 + 1) \approx 0{,}415$$

As can be seen from the results the tracker has high recall values, but low precision due to a large number of false positive detections. The tracker also often detects multiple smaller movements where there is actually one long movement (shown in Figure 8). This is due to the tracked object's velocity dropping to very low values during some parts of the movement.

It must be noted that the manual annotations contained only data about hand movements that were deemed communicatively significant. Very small movements and those caused by the rest of the body moving were not annotated. Therefore in reality the number of false positives can be slightly lower than represented by the evaluation results.

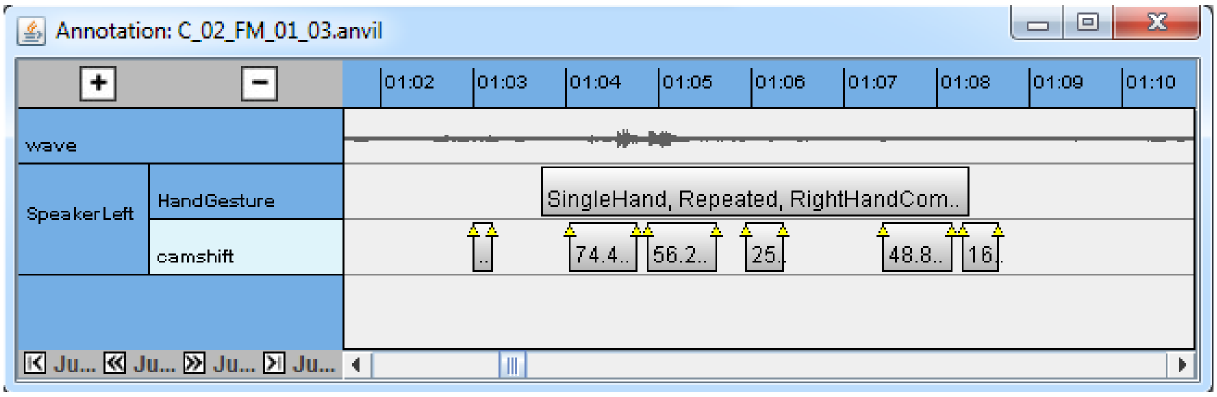


Figure 8 – Comparison of manual and tracker annotations

5 Discussion and Future Work

As mentioned in the previous section a big problem with tracking bare hands are other objects in the video that have a similar hue. This could be somewhat improved by increasing the saturation threshold. Unfortunately it appears that the skin hue saturation in the videos is also quite low so increasing the threshold would not completely absolve the problem.

The hand regions in the videos used for testing are quite small and not rectangular, which makes it more difficult to select a 'good' region for the initial hue values template. If a very small region is selected then the range of possible skin hue pixel representations is very small so it doesn't accurately represent the entire hand. If the region selected is bigger than the hand then background hue information will make the template inaccurate. A possible solution would be to allow the user to freely select the initial tracking region instead of using a rectangle. Obviously having a larger hand region would also provide a solution.

One of the biggest problems with using the tracker is that it detects movement when there is none. Looking at the back-projection images for the video shows that the probability values for static background objects change during the video. For non-static objects the changes are even larger, which makes the tracker move the search window and give the impression of movement. This is most likely caused by changes in lighting and the use of the lower-quality videos for testing. Increasing the movement detection threshold would solve this problem, but then actual movements that are below the threshold (for example very slow movements) would also not be detected. The movement detection algorithm would have to be improved so that it would ignore very small movements yet be able to recognise long and slow movements as a single gesture.

In order to improve the accuracy of tracking bare hands in lower quality videos options to better isolate the tracked object from its background and distractors present in the video should be researched. Preprocessing the video material in order to improve quality could also be considered.

In summary most of the problems could be solved by using high quality videos and improving the movement detection algorithm. Currently the pixel rate of the high quality raw MINT videos is impractical for tracking with the CAMShift algorithm, and so a trade-off is needed between the tracking accuracy and video quality. Due to the high number of false positive movement detections the tool does not provide a complete alternative to manual annotation, but could rather be used as a basis for determining where in a video hand gestures can be detected.

In the future a hand detection method could be added to the plug-in to reduce the chance of losing the search window during tracking and to remove the need for the user to manually locate the initial object region. An automatic gesture classification system could also be based on the current implementation.

Summary

The goal of this thesis was to describe various object tracking methods and to create a tool for automatic gesture annotation. Annotation of video data is an important prerequisite for human communication studies, but doing this manually is time and resource consuming. It is thus important to study automatic tools for annotation.

The tool implements an object tracking algorithm known as CAMShift and is used as a plug-in for the ANVIL annotation software. The tool is able to track hands and other coloured objects in a video and detect movements, however the inital detection of the hand is left to the user. The movements are automatically annotated by writing the start and end point of the movement, movement distance and average velocity to a specified annotation track in ANVIL.

The tool was tested on recordings of actual dialogues and used to track both bare hands and coloured objects. The results were evaluated by comparing annotations created by the tracker and manual annotations. Evaluation showed that the tracker is able to detect most hand gestures, but also produces a lot of false positive detections. The tracking and movement detection precision depends on the quality of the video being used and on the user specified settings.

All in all the created tool meets the goal of the thesis as it is able to automatically track and annotate gestures in recorded video conversations, however tracking bare hands in the evaluation videos proved to be too inaccurate. Thus the tool does not provide a complete alternative to manual annotation, but could rather be used as a basis for determining where in a video hand gestures can be detected. For further evaluation the tool would have to be tested on higher quality videos with more suitable lighting conditions.

Adding functionality to automatically detect hands in a video frame without the need for user intervention and classify gestures based on collected movement data would further reduce the need for user input during the annotation process. Possibilities to improve the accuracey of bare hand tracking would also have to be researched.

Käeliigutuste tuvastamine ja jälgimine videosalvestistes

Bakalaureusetöö (6 EAP)

Pihel Saatmann

Resümee

Antud lõputöö eesmärgiks oli kirjeldada erinevaid meetodeid objektide jälgimiseks videosalvestistes ning luua tööriist automaatseks käeliigutuste annoteerimiseks. Videosalvestiste annoteerimine on oluline vahend inimestevahelise suhtluse uurimiseks. Käsitsi annotatsioonide tegemine on aeganõudev ning seega on oluline uurida võimalusi automatiseeritud vahendite loomiseks.

Loodud tööriist kasutab *CAMShift* jälgimisalgoritmi ning on realiseeritud lisamoodulina programmile ANVIL. ANVIL on vabavaraline vahend annotatsioonide loomiseks. Tööriist suudab jälgida käsi ja värvilisi objekte ning tuvastada liigutusi videosalvestistes. Algne käepiirkonna või muu objekti videost ülesleidmine ja ära märkimine on jäetud kasutaja hooleks. Liigutused annoteeritakse automaatselt ning info liigutuse alg- ning lõpppunkti, keskmise kiiruse ning läbitud teekonna kohta kirjutatakse ANVIL'i annotatsioonifaili.

Loodud tööriista testiti videosalvestiste peal kahe inimese vahelisest suhtlusest ning kasutati käte ning muude värviliste objektide jälgimiseks. Tulemusi võrreldi käsitsi tehtud annotatsioonidega ning selle põhjal hinnati tööriista täpsust liigutuste tuvastamisel. Hindamisel ilmnes, et jälgija suudab tuvastada suurema osa käeliigutustest, kuid tuvastab ka palju vale-positiivseid liigutusi. Jälgimise ja liigutuste tuvastamise täpsus sõltub videokvaliteedist ning kasutaja poolt määratud sätetest (näiteks minimaalne arvestatav värviküllastus). Testimisel ilmnes, et paljaste käte jälgimine on problemaatiline ning seega tuleks tööriista testida parema kvaliteediga videote peal ning uurida võimalusi jälgimistäpsuse parandamiseks, kui videokvaliteet on halb.

Kokkuvõtteks võib öelda, et loodud tööriist täidab seatud eesmärke, kuid on ruumi täiendusteks. Näiteks võiks lisada võimalused automaatseks käepiirkondade leidmiseks videos ning liigutuste analüüsimiseks ja eri kategooriatesse jagamiseks. Sellised täiendused vähendaksid veel rohkem kasutajalt nõutavat tööd. Antud seisus on tööriist mõeldud pigem manuaalse annoteerimisprotsessi lihtsustamiseks kui selle täielikuks alternatiiviks.

References

[1] Kipp, M. (2001) Anvil - A Generic Annotation Tool for Multimodal Dialogue. Proceedings of the 7th European Conference on Speech Communication and Technology (Eurospeech), pp. 1367-1370.

[2] Jokinen, K. and S. Tenjes (2012). 'Investigating Engagement - intercultural and technological aspects of the collection, analysis, and use of the Estonian Multiparty Conversational video data'. In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), 23-25 May 2012, Istanbul, Turkey

[3] Yilmaz, A., Javed, O., and Shah, M. (2006). 'Object tracking: A survey'. *ACM Computing Surveys (CSUR)*. 38 (4), Article 13. Available at: http://crcv.ucf.edu/papers/Object%20Tracking.pdf (Accessed: 11th May 2014)

[4] Han, Bing; Paulson, Christopher; Lu, Taoran; Wu, Dapeng; Li, Jian (2009). 'Tracking of Multiple Objects under Partial Occlusion'. *Automatic target Recognition XIX,* 7335. Available at: http://www.wu.ece.ufl.edu/mypapers/trackingSPIE09.pdf (Accessed: 11th May 2014).

[5] Viola, P., Jones, M. J. (2004). 'Robust Real-Time Face Detection'. *International Journal of Computer Vision*. 57 (2), pp.137–154.

[6] Bradski, G. R. (1998). Computer video face tracking for use in a perceptual user interface. *Intel Technology Journal*. Q2, pp.705-740.

[7] Hewitt, R. (2007) 'Seeing With OpenCV, Part 3: Follow that Face!', *SERVO Magazine.* [Online]. Available at: http://www.cognotics.com/opencv/servo_2007_series/part_3/index.html (Accessed: 11th May 2014).

[8] Laganière, R (2011). *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing Ltd, Olton Birmingham, GBR. pp.90-94. [Online] Available at: http://my.safaribooksonline.com/9781849513241/91 (Accessed: 11th May 2014).

[9] HSV colour model. [Online] Available at: http://www.ncsu.edu/scivis/lessons/colormodels/color_models2.html (Accessed: 11th May 2014).

[10] Bart Jongejan (2012). Automatic annotation of head velocity and acceleration in Anvil. In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), 23-25 May 2012, Istanbul, Turkey.

[11] OpenCV (Open Source Computer Vision) Wiki. Available at: http://code.opencv.org/projects/opencv/wiki (Accessed: 11th May 2014).

[12] Audet, S. (2013). JavaCV. Available at: https://github.com/bytedeco/javacv (Accessed:

11th May 2014).
[13] Bradski, G.R and Kaehler, A. (2008) *Learning OpenCV.* O'Reilly Media, Sebastopol. pp.42, 341.

Appendix 1 – Starting the Plug-in

Install ANVIL and extract the plug-in files to the installation directory (detailed installation guide are in the INSTALLATION_GUIDE.txt file that is included with the installation files, see Appendix 2). Start ANVIL using the ANVIL_CAMSHIFT_TRACKER.BAT file and follow these steps:

1. Add the plug-in to ANVIL:
   - Edit → Options → Plug-ins → Add.
     - The title can be anything.
     - The class has to be AnvilHandTracker.

Steps 2, 3 and 4 may be done in any order.

2. Create a specification:
   - Edit → Edit specification.
   - Either create a new one or load an existing one.
   - Add track/group → Add track.
     - Name can be anything, type should be primary.
   - Add attribute (optional).
     - Name should be “movement”, type TimeStamped Point.
   - Add attribute (optional).
     - Name should be “velocity”, type String.
   - Add attribute (optional).
     - Name should be “distance”, type String.

3. Start the plug-in:
   - Tools → [Name of the plug-in]

4. Load a video or annotation:
   - File → Open → [Video or annotation] → Browse specification → [The specification you created earlier].

5. Select the correct track in the annotation window by clicking on it. If the annotation window is blank then try resizing it.

6. Start the tracker by selecting an initial search window in the main video window and clicking Start in the tracker controls.
   - If the selected track doesn't have suitable movement and velocity attributes an error message is displayed in the console and tracking won't start.
   - If selecting the search window doesn't work then try to switch to another track and back again (it is important to select the track after starting the plug-in!)

Appendix 2 – Additional files

The MINT video files used for testing, annotation data created with the tracker and installation files can be found at https://www.dropbox.com/sh/oedg3fsanatye73/AABcGE9HqTNEi4wOUrIHf3Kma
The source code and installation files for the tracker are also included with this thesis as an archive file.