3D Face Tracking Using Stereo Cameras with Whole Body View

Maria Mikhisor

a thesis submitted for the degree of

Doctor of Philosophy

at the University of Otago, Dunedin,

New Zealand.

7 July 2017

Abstract

All visual tracking tasks associated with people tracking are in a great demand for modern applications dedicated to make human life easier and safer. In this thesis, a special case of people tracking - 3D face tracking in whole body view video is explored. Whole body view video means that the tracked face typically occupies not more than $5 - 10\%$ of the frame area. Currently there is no reliable tracker that can track a face in long-term whole body view videos with luminance cameras in the 3D space.

I followed a non-classical approach to designing a 3D tracker: first a 2D face tracking algorithm was developed in one view and then extended into stereo tracking. I recorded and annotated my own extensive dataset specifically for 2D face tracking in whole body view video and evaluated 17 state of the art 2D tracking algorithms. Based on the TLD tracker, I developed a face adapted median flow tracker that shows superior results compared to state of the art generic trackers. I explored different ways of extending 2D tracking into 3D and developed a method of using the epipolar constraint to check consistency of 3D tracking results. This method allows to detect tracking failures early and improves overall 3D tracking accuracy. I demonstrated how a Kinect based method can be compared to visual tracking methods and compared four different visual tracking methods running on low resolution fisheye stereo video and the Kinect face tracking application.

My main contributions are:

- I developed a face adaptation of generic trackers that improves tracking performance in long-term whole body view videos.
- I designed a method of using the epipolar constraint to check consistency of 3D tracking results.

Acknowledgements

I would like to thank my supervisors Geoff Wyvill, Brendan McCane and Steven Mills. This thesis would not have been possible without their guidance and help.

I must thank Holger Regenbrecht and Tobias Langlotz from the Information Science department for their much valued contributions to parts of this research.

I would like to thank all the wonderful people at the Computer Science department for making my studies at the University of Otago such a rewarding experience: Hamza Bennani, Jordan Campbell, Leila Eskandari, Aleksei Fedorov, Claudia Ott, Allan Hayes, Paul Crane and many many others.

Most importantly, I want to express my gratitude to my family for endless support throughout my whole studies. My final and very special thanks goes to my baby Leo for his patience and for making me a more organized student and to my husband Maxim, for his support, comprehension and encouragement.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Requirements analysis . . . 4
- 1.3 Contributions . . . 7
- 1.4 Thesis layout . . . 8

2 Literature Survey **10**
- 2.1 3D tracking . . . 10
  - 2.1.1 Cameras number and position . . . 11
  - 2.1.2 Using stereo information . . . 14
  - 2.1.3 Foreground segmentation . . . 17
  - 2.1.4 Tracking Space . . . 18
  - 2.1.5 3D tracking without building a depth map . . . 25
  - 2.1.6 Tracking methods . . . 26
  - 2.1.7 3D tracking review summary . . . 29
- 2.2 2D face tracking . . . 32
- 2.3 Conclusion . . . 35

3 Tracking evaluation framework **36**
- 3.1 Collecting evaluation datasets . . . 36
- 3.2 Stereo setup and calibration . . . 39
- 3.3 Ground truth annotation . . . 43
- 3.4 Tracking evaluation . . . 45
  - 3.4.1 Evaluation metric for 2D tracking . . . 45
  - 3.4.2 Evaluation metric for 3D tracking . . . 48
  - 3.4.3 Frame rate estimation . . . 51
- 3.5 Conclusion . . . 53

4 2D Face Tracking **55**
- 4.1 Choosing the best public tracker . . . 55
- 4.2 Speeding up TLD . . . 57
  - 4.2.1 Background subtraction . . . 58
  - 4.2.2 Subwindow detection . . . 59
- 4.3 Adapting for face tracking . . . 61
  - 4.3.1 FaceTLD . . . 62
  - 4.3.2 Using structural constraints for creating the face model . . . 64

4.3.3 Face adaptation implementation . . . 66
4.3.4 Adapting generic trackers for tracking faces . . . 73
4.3.5 Different modifications of TLD . . . 78
4.4 Experimenting with parameters of face adapted median flow . . . 80
4.4.1 Frequency of running the face detector . . . 80
4.4.2 Subwindow based detection . . . 82
4.4.3 Different sliding window methods . . . 84
4.4.4 The "following" subwindow size . . . 86
4.4.5 Sliding window size . . . 88
4.4.6 Average face size and Haar cascade limits . . . 89
4.4.7 Final comparison: TLD modifications and context tracker . . . 91
4.5 Conclusion . . . 92

5 3D Face Tracking **94**
5.1 Converting 2D tracking into 3D tracking by triangulation . . . 94
5.2 Using stereo information . . . 96
5.2.1 Checking size and 3D position . . . 97
5.2.2 Sharing information between the views . . . 100
5.3 3D median flow tracker . . . 102
5.3.1 3D Lucas-Kanade tracking . . . 102
5.3.2 Two head models and two ways of computing displacement . . . 105
5.3.3 Forward-backward tracking . . . 108
5.3.4 Comparing 2D and 3D median flow trackers . . . 111
5.4 Compliance with requirements . . . 112
5.5 Conclusion . . . 113

6 3D Face tracking in fisheye video **114**
6.1 Comparing Kinect to visual tracking methods . . . 116
6.2 Using a particle filter for 3D tracking . . . 118
6.3 Comparing 3D trackers on fisheye video and Kinect face tracking . . . 123
6.4 Conclusion . . . 126

7 Conclusion **127**
7.1 Future work . . . 128

References **130**

List of Tables

3.1 The table describes common challenging factors and their occurrence in my evaluation datasets. 38

4.1 Average overlap of the face adapted TLD tracking results and the ground truth boxes for different values of *threshold_1*. In bold is the default value that was used in all other experiments. 71

4.2 Average overlap (**AO**) of the face adapted TLD tracking results and the ground truth data for different values of thresholds used in Alg. 3. The values used as default are shown in bold for each threshold. 74

6.1 Kinect field of view. 123

List of Figures

1.1 The concept of the off-axis projection. The image is taken from Oygard (2012) with permission from the author. 3
1.2 Left: Standard display; distortion from oblique viewing and no viewpoint adoption. Right: The proposed display system; no distortion and passive viewpoint selection through user gaze tracking conveys the illusion of 3D content. The image and caption are taken from Malleson and Collomosse (2013) with permission from Springer. 3
1.3 Challenges in 3D face tracking: scale change. 5
1.4 Challenges in 3D face tracking: cluttered background. 6
1.5 Challenges in 3D face tracking: change in illumination. 6

2.1 An example of a wide baseline system with four cameras. Four views from different perspectives allow to separate 9 people, even though they occlude each other in each view. The image and caption are taken from Khan and Shah (2006) with permission of Springer. 12
2.2 A filtered image of the tracked person's head and shoulders extracted from the depth map is compared to the head and shoulders template. This image is taken from Choi *et al.* (2011) ©IEEE. 13
2.3 The top corner position is taken from Bahadori *et al.* (2007) with permission of Springer. The top down position is taken from Englebienne *et al.* (2009) ©IEEE. The front facing position is taken from Muñoz-Salinas *et al.* (2007) with permission from Elsevier. 14
2.4 Schematic representation of stereo cameras setup. 15
2.5 The foreground segmentation from a disparity map. (a) One of the two luminance images, (b) raw disparity map after morphological smoothing, (c) regions of slowly varying disparity, and (d) silhouettes recovered after connected components grouping. The image and caption are taken from Darrell *et al.* (2000) with permission of Springer. 17
2.6 Segmenting the height map background model. The upper row shows frames in different time instances. The middle row depicts height maps in the same time instances. In the lower row the median of the height maps for each time instance is shown. In this row it is visible, how the foreground is gradually filtered out. The image is taken from Muñoz-Salinas (2007) with permission from Elsevier. 19
2.7 Tracking body parts in the depth map space (Nanda and Fujimura, 2004) ©IEEE. Tracking (a) hand, (b) body torso and (c) head. 20

2.8 (a) Luminance channel, (b) corresponding disparity map, (c) 2D red channel-disparity histogram, (d) 2D green channel-disparity histogram, (e) 2D blue channel-disparity histogram. Image and its caption are taken from Zoidi *et al.* (2014) with permission from Elsevier. 21
2.9 Generating a plan-view height map and occupancy map. Image is taken from Harville and Dalong Li (2004) ©IEEE. 22
2.10 (a)Camera image. (b) 3D reconstruction of the scene. (c) Occupancy map. (d) Height map. (e) Coloured height map. The image is taken from Muñoz-Salinas (2008) with permission from Elsevier. 23
2.11 Observations consist of two maps. (a) A simpl binary plan-view image of foreground points. (b) A map of audio power in the microphone array coordinate system. Image is taken from Checka *et al.* (2003) ©IEEE. . 23
2.12 (a) Feature points extracted from the original grey scale image and (b) their projections to the world coordinate system. The green polygon is the region of interest used to remove background points. The image is taken from Cai *et al.* (2010) with permission from Elsevier. 26
2.13 The concept of region based stereo matching. The point of intersection of the quadrilateral formed by back-projecting the end-points of the matched segments yields a 3D point lying inside an object. The matching segments are 1 and 1', and 2 and 2' respectively. This image and its caption are taken from Mittal and Davis (2003) with permission of Springer. 27
2.14 Using three models with different life spans for face tracking. Image is taken from Li *et al.* (2008) ©IEEE. 34
2.15 Visual constraints for face tracking used in the work of Kim *et al.* (2008). (a) A generative PCA model where each row represents a certain pose subspace. To predict a pose I_t, the shortest distance between the current tracked face I_t and all the subspaces is found. (b) SVM classifier that discriminates face-centred images (+, in blue) against badly cropped face images (−, in red). This image was taken from Kim *et al.* (2008) ©IEEE. 34

3.1 Examples from the 2D dataset with public videos. 40
3.2 Examples from my high resolution gray scale stereo dataset. 40
3.3 Examples from my fisheye color stereo dataset. 40
3.4 The shared viewing volume of the stereo system. 42
3.5 Different ground truth labelling formats: axis aligned bounding box, oriented bounding box, head contour. 43
3.6 If the tracker outputs the target bounding box, when the target is not really visible, or the tracker cannot find the target, when the target is present in the image, then the penalty overlap and penalty location error are added to the final result. 47
3.7 The relative 3D error is a ratio of the absolute location error e and the distance between the true target position and the midpoint of the cameras d. 49

3.8 Three different location error plots (the 2D location error plot and the 3D location error plot in the top row, the relative error plot in the bottom row) show the results of the experiment that compares 3D face adapted TLD and 3D face adapted median flow. This experiment is described in Section 5.1. 51
3.9 In the face adapted median flow tracker, the frame rate is slower for the frames when the target is lost and the whole frame detection is performed to restart the tracker. 52
3.10 The successful tracking frame rate and the average frame rate for face adapted TLD for 37 high resolution videos. The two frame rates are very similar in this experiment. 53

4.1 The performance of the generic tracking algorithms on our dataset. . . 56
4.2 The performance of the generic tracking algorithms on the public face tracking dataset. 56
4.3 Background subtraction. The white rectangles show the segmented foreground. The yellow rectangle shows the tracked target. 58
4.4 The performance of TLD with background subtraction and without it. 58
4.5 The subwindow detection. Red boxes show the detection area. Yellow solid line boxes show the tracked target. The dashed yellow boxes show the false positive examples. 60
4.6 The performance of TLD and FastTLD on high resolution sequences and on low resolution sequences. 61
4.7 The frame rates of TLD and FastTLD in low resolution images and in high resolution images. 62
4.8 The Viola Jones face detector finds many false positive examples. . . . 63
4.9 Illustration of a scanning grid applied to three consecutive frames (a) and corresponding spatio-temporal volume of labels with unacceptable (b) and acceptable (c) labelling. Red dots correspond to positive labels. The image is taken from Kalal (2011). 65
4.10 Illustration of a trajectory in video volume and corresponding trajectory in the appearance space. 66
4.11 The block diagram of the face tracking adaptation method. 67
4.12 When tracking and detection is successful, the structural constraints are applied to the results. 70
4.13 If the tracker and the detector results overlap is lower than a threshold, the tracker is reinitialized. 71
4.14 If the tracker and the detector results do not overlap, the tracker needs to be reinitialized. To reinitialize the tracker, the detection result with the highest similarity to the face model needs to hold for at least two frames. 73
4.15 Performance evaluation on public image sequences. The overlap success plots for the three trackers are shown in the top row and the location precision plots are shown in the bottom row. Each line in a plot shows the mean performance over the image sequences and the shading shows one standard deviation. 75

4.16 Performance evaluation on my image sequences. The overlap success plots for the three trackers are shown in the top row and the location precision plots are shown in the bottom row. Each line in a plot shows the mean performance over the image sequences and the shading shows one standard deviation. 76
4.17 Examples of tracking results highlighting how face tracking adaptation helps to recover tracking after occlusions or rapid movements. The red boxes show the original trackers' performance and the green boxes show the face tracking adaptation performance. 77
4.18 The comparison of TLD, Struck and MIL, adapted for tracking faces. . 77
4.19 The performance of TLD, face adapted TLD, face adapted median flow. 79
4.20 The frame rate summary for the trackers. 79
4.21 TLD retains the shape of the initial bounding box (green rectangle). If it is reinitialized by the face detector, the shape of the bounding box is changed to a square (yellow rectangle). 80
4.22 The performance of the median flow tracker adapted for tracking faces with different frequencies of running the face detector. 81
4.23 The frame rate summary for the median flow with different detection frequencies. 81
4.24 The face detection is performed in two subwindows. Subwindow 1 is a subwindow centred at the previous target position. Subwindow 2 is positioned in the background. It this example two faces are found. They are classified as either positive or negative examples in the next step. . 83
4.25 The performance of face adapted median flow with detection performed on a full frame once every 10 frames and face adapted median flow with subwindow based detection. 83
4.26 The frame rates of face adapted median flow with detection performed on a full frame once every 10 frames and face adapted median flow with subwindow based detection. 84
4.27 The illustration of the different sliding window methods. 85
4.28 The performance of face adapted median flow with different sliding window methods 86
4.29 The frame rate of face adapted median flow with different sliding window methods. 87
4.30 The performance of face adapted median flow with different sizes of the "following" subwindow. 87
4.31 The frame rate of face adapted median flow with different sizes of the "following" subwindow. 88
4.32 The accuracy of face adapted median flow with different sizes of the sliding window: 0.1, 0.2, 0.3, 0.4, and 0.5 of the frame height. 88
4.33 The frame rate of face adapted median flow with different sizes of the sliding window. 89
4.34 The face height distribution in the high resolution videos in our dataset. 90
4.35 The performance of different detection limits in the high resolution videos. 90
4.36 The frame rate for different detection limits. 91

4.37 The final comparison of face adapted median flow, TLD, face adapted TLD and CXT. . . . 91
4.38 The frame rate comparison for face adapted median flow, TLD, face adapted TLD and CXT. . . . 92

5.1 The block scheme of a simple 3D tracker composed out of two independent instances of a 2D tracker. This way any 2D tracker can be converted into 3D. . . . 95
5.2 The location error plot for 3D face adapted median flow and 3D TLD. The success ratio at the threshold of 20 cm is given in brackets after the trackers names. . . . 95
5.3 The frame rate plot for the 2D and 3D face adapted median flow and 2D and 3D TLD. . . . 96
5.4 The illustration of the epipolar constraint. The face is denoted by the bounding box in the left image. In the right image the epipolar line corresponding to the center of the bounding box is shown. . . . 97
5.5 Checking the consistency of the 3D result. . . . 98
5.6 The location error plot for 3D face adapted median flow with the size and 3D position check and without it. . . . 99
5.7 The location error plot for different overlap thresholds. . . . 100
5.8 The collections of positive and negative examples are shared between the views. . . . 101
5.9 The location error plot for 3D face adapted median flow with shared face examples collections and with separate collections. . . . 101
5.10 The 3D point $\mathbf{p}$ in the C system is projected to the 3D point $\mathbf{p}'$ in the C' coordinate system. . . . 103
5.11 The set of points in the cylinder head model. . . . 106
5.12 Angle θ between the surface normal and the direction from the head center to the camera center, which is used to determine the pixel density weight. The image is taken from Xiao *et al.* (2002) ©IEEE 107
5.13 The set of points on the flat rectangle face model. . . . 108
5.14 A separate displacement is computed for each sample point. . . . 108
5.15 The location error plot for the 3D Lucas-Kanade tracker with flat model and the cylinder model. . . . 109
5.16 The FB error penalizes inconsistent trajectories. Point 1 is visible in both images, tracker works consistently forward and backward. Point 2 is occluded in the second image, forward and backward trajectories are inconsistent. The image is taken from Kalal *et al.* (2010b) ©IEEE. . . . 109
5.17 The Lucas-Kanade tracker accepts a bounding box and a pair of images. A number of points within the bounding box are tracked, their error is estimated and the outliers are filtered out. The remaining estimate the bounding box motion. The image is taken from Kalal *et al.* (2010b) ©IEEE. . . . 110
5.18 The location error plot for the 3D Lucas-Kanade tracker with FB tracking and without it. . . . 110
5.19 The location error plot for the 3D median flow and the 2D median flow. 112

5.20 The frame rate plot for 3D median flow and 2D median flow. 112

6.1 Top row: the joint viewing zone and an image example for the cameras with perspective lenses. Bottom row: the joint viewing zone and an image example for the cameras with fisheye lenses. 115
6.2 Selecting corresponding points in the depth sensor image and the fisheye camera image. The three corresponding points (the head and hands of the wooden figure) are marked by red circles in both views. 117
6.3 A synchronization sequence was recorded in the start and end of each evaluation video to help synchronizing the Kinect data and the fisheye video. 117
6.4 Three stages of the particle filter framework. The image is taken from Li *et al.* (2016). 120
6.5 For each 3D particle, a 3D position is projected to the image plane and then an ellipsoid of the size that corresponds to 0.2 meters in 3D world, is created. 122
6.6 A visualization of 100 particles, projected to one of the cameras view. Ellipses with lighter colour have higher likelihood. Black ellipses have very small likelihood. 122
6.7 The location error plot for the five 3D trackers evaluated on the fisheye stereo dataset in the unconstrained viewing zone. 124
6.8 The location error plot for the five 3D trackers evaluated on the fisheye stereo dataset in the Kinect viewing zone. 124
6.9 The frame rate plot for the five 3D trackers evaluated on the fisheye stereo dataset. 125

Chapter 1

Introduction

Visual tracking is a diverse research area that has always attracted a lot of attention and has multiple applications. All tracking tasks associated with people tracking are in great demand for modern applications dedicated to make human life easier and safer. Human silhouette tracking is often used in surveillance (Cai *et al.*, 2010), hands are tracked for gesture recognition (Rautaray and Agrawal, 2015), gaze tracking is used in hands free displays (Sippl *et al.* 2010, Zhang *et al.* 2013), face tracking is used in personality (Abate *et al.*, 2007) and emotion recognition (Pantic and Bartlett 2007, Wang *et al.* 2006), etc. In this thesis, I explore a special case of people tracking - 3D face tracking in whole body view video.

1.1 Motivation

In this research I explore long-term face tracking in whole body view videos. Whole body view tracking means that the camera can see a wide area, and a person can move freely in front of the camera. The tracked face in this scenario typically occupies less than $5-10\%$ of the frame area, but can come up to $50-60\%$. This type of tracking scenarios is less constrained compared to the popular head-and-shoulders videos when a person sits in front of a desktop camera and the face and upper body occupies most of a frame. Whole body view videos often feature a large scale change, fast motion, occlusion, lighting change and other challenging factors. In this type of scenarios cameras are usually mounted on a wall in a room, auditorium, street or some other public space. Long-term unconstrained tracking implies that a person can leave the field of view and then come back, and the tracking algorithm needs to be able to reacquire the tracked target.

This type of face tracking algorithms can be effectively used in interactive displays such as public displays and personal 3D interactive displays. The falling cost of display hardware has led to a proliferation of screens of varying sizes, shapes, and forms in public spaces. Displays are used for advertising or as digital signage providing electronic equivalents of paper advertising and information boards (Davies, Nigel, Langheinrich, Marc, Krüger 2016, Davies *et al.* 2012). However, the vast majority of today's public displays effectively disappear: people have become so accustomed to their low utility that they are highly skilled at ignoring them (Muller *et al.*, 2009). As existing digital signage systems evolve towards more sophisticated pervasive display networks, the introduction of personalized, interactive content on public displays could engage viewers and promote social interaction like never before. Introducing 3D face tracking into public displays allows us to track people looking at the display and interact with them by changing the displayed content dynamically according to a viewer's position. This can help to grab attention of people passing by the display and make their attention span longer.

Despite the advantages of viewer tracking, very few research works feature public displays that use any kind of user tracking. Close range gaze tracking (0.5-3 meters in front of a display) that allows hands free control of the content is the most popular approach (Zhang *et al.* 2015, Zhang *et al.* 2013, Nakanishi *et al.* 2002, Sippl *et al.* 2010 and other works). Ren *et al.* (2013) perform gesture recognition to control public displays also in the near range. In all these works, 3D head position is not estimated.

Personal 3D interactive displays also known as fish tank virtual reality displays are another application of 3D face tracking. Such displays can be used for gaming or for practical applications such as art design or engineering. De Boer and Verbeek (2009) propose to use this type of displays as windows in windowless environments. Fish tank VR displays use motion parallax as the main depth perception cue. To represent the motion parallax depth cue, head tracking is performed and the off-axis projection matrix is dynamically updated according to the viewer's head position[1]. The off-axis projection allows to counteract the distortion that arises when the user is looking at the screen from an angle. The off-axis projection concept is illustrated in Figure 1.1. In Figure 1.2 shows two screens, one with a classic on-axis projection and one with an off-axis projection.

Many fish tank VR displays described in the literature work in the range of 0.5 - 1.5

[1] A video example of a fish tank virtual reality display that I assembled using Kinect, can be found here https://youtu.be/0KMqxDySFA4

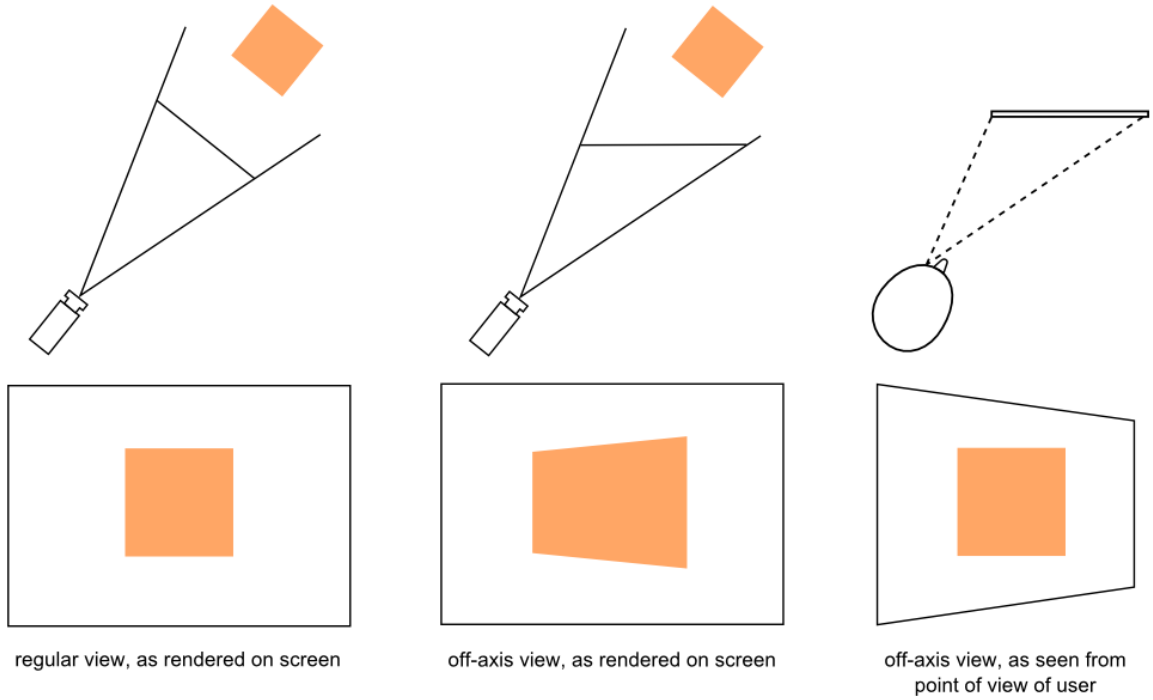

Figure 1.1: The concept of the off-axis projection. The image is taken from Oygard (2012) with permission from the author.

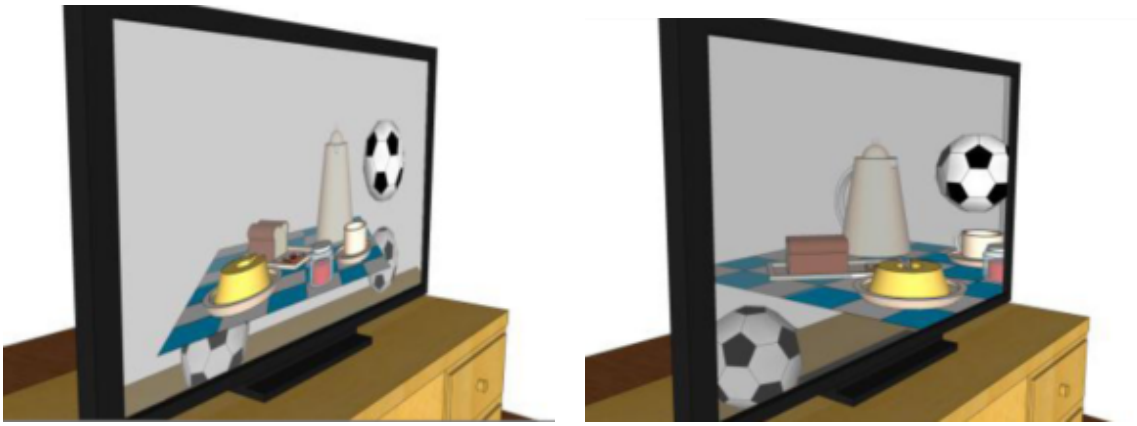

Figure 1.2: Left: Standard display; distortion from oblique viewing and no viewpoint adoption. Right: The proposed display system; no distortion and passive viewpoint selection through user gaze tracking conveys the illusion of 3D content. The image and caption are taken from Malleson and Collomosse (2013) with permission from Springer.

meters (for example Rekimoto and Building 1995, Lee *et al.* 2013, Surman *et al.* 2015). When using these displays the user is supposed to sit or stand close to the display and

face tracking is performed by a web camera. Brar *et al.* (2010) track the user's face in a larger zone (1-3 meters) for their head coupled display. In their work, face tracking is performed by 6 cameras using a face detector and block matching. Several authors use Kinect for 3D interactive displays (Dundas and Wagner 2014, Johnsen *et al.* 2013, Lee *et al.* 2013). Kinect allows a viewing zone of 1.3 - 3.5 meters and the viewing angle is limited by 70° horizontally and 60° vertically. Another downside of using Kinect is that its performance can be disturbed by sunlight. Another popular method is using active IR illumination and the bright pupil effect for pupil detection. Xue and Wang (2014) use IR pupils detection combined with a cascaded classifier for detecting faces and a support vector machine for detecting eyes. After eyes are detected, a Kalman filter is used to predict the eyes position in the next moment in the range of 60 - 75 cm from the display. Lee (2008) uses the Wii remote to track the user wearing glasses with two IR emitters. All these displays have a limited viewing zone that restricts their usage.

The challenges of 3D face tracking for interactive displays include a large scale change as the tracked person can be very far and then come close to the display. Interactive displays are often used in public spaces and this introduces such tracking difficulties as a cluttered background and other people faces that can interfere with the target face tracking. Tracking has to be done in real time to allow smooth interaction with a display and engaging experience for the user. Another important factor is that a person can leave the field of view and then come back, and the tracking algorithm needs to be able to reacquire the tracked face.

3D face tracking is not limited to interactive displays. For example, face tracking can be used to track a speaker or a lecturer to provide an automatic close view of their face when recording a public speech (Nickel and Gehrig, 2005). Even though there are multiple possible applications, the goal of this thesis is rather theoretical. The task of whole body view face tracking is not well studied and described in the literature (see Chapter 2). I would like to address this gap.

1.2 Requirements analysis

The goal of thesis is to develop a 3D face tracking algorithm to track a person moving freely (walking, running, jumping, etc.) in front of stereo cameras. Tracking is active when the user looks towards the cameras. I set the following requirements for the final 3D face tracking algorithm:

- **Using luminance cameras only**. It is possible to solve this research goal partially using such devices as Kinect. However, I chose to use luminance cameras only and no other additional sensors such as depth or audio sensors.

- **Viewing range 0.5 - 5 meters**. This range is chosen as an average size of a public indoor environment such as an exhibition room or a classroom. The user of the system should be able to move freely in front of the cameras, come close and go as far as 4-5 meters away. This results in significant scale change. See Figure 1.3 for an illustration of this issue.

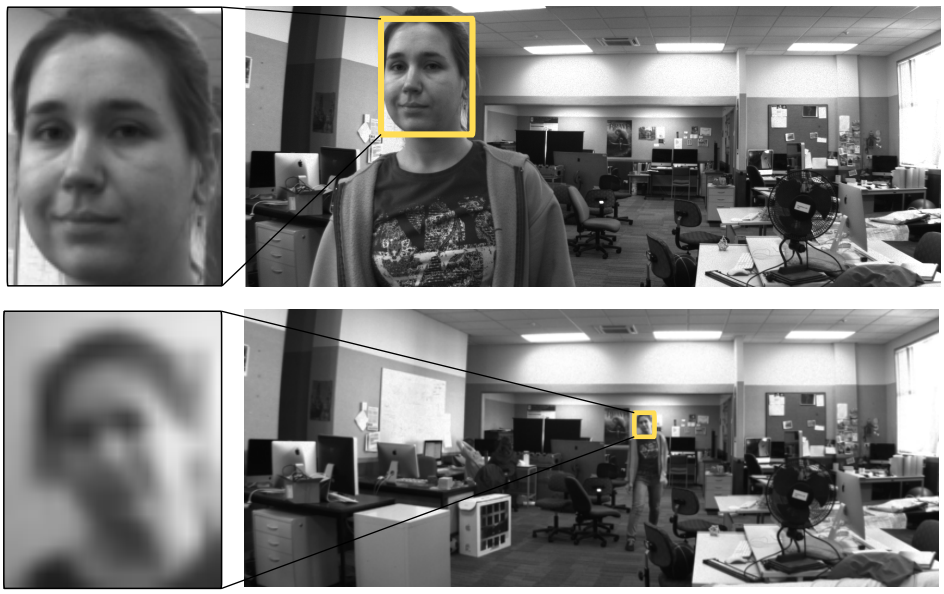

Figure 1.3: Challenges in 3D face tracking: scale change.

- **Frame rate close to 30 frames per second**. The final algorithm should be able to work in real time. 30 frames per second is considered to be fast enough for real time processing. The exact frame rate needed for tracking in a real time application depends on the specifics of this application, such as how much post-processing is needed after finding the face and how much time lag can go unnoticed by the user.

- **Long term tracking**. The tracked person can leave the viewing zone and return after some time. The tracker must resume tracking when the person returns.

- **Indoor environment**. I chose to develop the 3D face tracking algorithm for indoor environments. It does not mean that it will not work outside. However, all evaluation datasets are recorded indoor.

- **Cluttered background and occlusions**. Other people can be present in the scene. The tracker should be able to distinguish the tracked person from other people and from background objects. It should be able to recover quickly from occlusions. See Figure 1.4 for an illustration of confusing background.

Figure 1.4: Challenges in 3D face tracking: cluttered background.

- **Change in illumination**. Drastic lighting variations can affect the tracked person's appearance. The tracking algorithm should be able to cope with these effects. See Figure 1.5 for an illustration of lighting variations.

Figure 1.5: Challenges in 3D face tracking: change in illumination.

1.3 Contributions

I followed a non-classical approach of developing a 3D tracking algorithm: instead of using stereo matching techniques, I developed a 2D face tracking algorithm in one view and then extended it into stereo tracking. First, I compared state of the art 2D trackers to find the one that shows the best performance for long term face tracking with a large scale change. After that I improved the chosen tracker accuracy by adapting it for tracking exclusively faces. Lastly, I explored different ways of extending this 2D face tracker into 3D tracking. The following contributions were made in this research:

- I collected and labelled a dataset of more than 100 stereo and monocular view videos of different resolution and complexity specifically for evaluating long term whole body view face tracking. All videos are 600-3000 frames long.

- I evaluated 17 state of the art 2D trackers and found that TLD (Kalal *et al.*, 2011) and the context tracker (Dinh *et al.*, 2011) show the best performance for long term whole body view face tracking.

- I designed and implemented an adaptation of generic trackers for tracking exclusively faces. This adaptation method employs the Viola Jones face detector and structural constraints to collect a set of positive and negative examples of the tracked face and help to restart the tracker when it fails. Trackers adapted for face tracking show considerable accuracy improvement over generic trackers. This contribution was published in Mikhisor *et al.* (2015).

- I developed a close to real time face adaptation of the median flow tracker (Kalal *et al.*, 2010b). Originally, the median flow tracker is a part of TLD. However, my experiments showed that face adapted median flow outperforms face adapted TLD and the context tracker for the face tracking task. Also, I designed a sliding window detection method that allows to run face adapted median flow close to real time on low resolution videos.

- I explored different ways of extending 2D face adapted median flow into 3D and developed a way to check 3D tracking results consistency using the epipolar constraint. For each pair of 2D face bounding boxes, the resulting 3D head is reconstructed and reprojected back into the images. If the tracking result is reliable, then the reprojected face boxes overlap with the 2D tracking boxes. This way the epipolar constraint helps to detect tracking failures early and improves overall 3D tracking accuracy.

- I demonstrated how a Kinect based tracking method can be compared to visual tracking methods running on luminance stereo video. To compare visual tracking methods to Kinect, the Kinect face tracking results have to be recorded at the same time as recording the dataset using luminance cameras. To be able to compare the tracking results, Kinect and the stereo cameras have to be synchronized and stereo calibrated.

- I compared the 3D Kinect face tracking application and four visual trackers running on low resolution fisheye stereo video: the 3D particle filter, two 2D particle filters with triangulation, 3D face adapted median flow, 3D face adapted TLD. The 3D particle filter shows the best performance. It outperforms Kinect because the Kinect's viewing zone is limited. It also outperforms 3D face adapted median flow and 3D face adapted TLD because these two trackers cannot track faces smaller than 10×10 pixels in low resolution fisheye videos. The 3D particle filter was published in Mikhisor *et al.* (2014).

1.4 Thesis layout

The rest of this thesis is structured as follows. Chapter 2 reviews state of the art algorithms for whole body view people tracking in 3D space. Works on face tracking are also reviewed in this chapter.

Chapter 3 describes the preparation steps required to design a tracking algorithm: stereo calibrating cameras, recording and annotating evaluation datasets, choosing metrics that will be used to compare different algorithms.

In Chapter 4 I explore the existing state of the art trackers to find out which one works best for tracking faces in whole body view videos. TLD and the context tracker show the best performance. Then I develop a novel approach of adapting generic trackers for tracking faces. Finally, face adapted median flow is developed which shows superior accuracy over face adapted TLD and runs close to real time on low resolution videos.

Chapter 5 focuses on converting a 2D tracking algorithm into 3D tracking. A novel method of checking the consistency of the target size and 3D position using the epipolar constraint is developed. This method helps to detect failures early and restart the tracker, but does not help to speed up the final 3D tracker. It can be used when converting any 2D tracker into 3D. In the previous chapter it is shown that the face adapted median flow tracker has an optimal combination of speed and accuracy for

real time tracking. The Lucas-Kanade tracker is one of the key parts of this tracker. In this chapter I also study different ways of converting the 2D Lucas-Kanade tracker into 3D to optimize its performance and speed up the final 3D tracker.

In Chapter 6 I investigate 3D face tracking in fisheye stereo video. Using fisheye cameras allows much wider field of view which can be very useful in many applications. For this dataset only, the Kinect 3D tracking data was recorded at the same time when recording this dataset. This allows me to compare Kinect 3D face tracking to 3D face adapted median flow and 3D face adapted TLD developed in the previous chapter. Also I evaluate two more trackers on the color fisheye dataset: a 3D particle filter and two 2D color particle filters with triangulation. These two methods were not evaluated in the previous chapter because they do not work on my main greyscale stereo dataset. The 3D particle filter shows good performance on the low resolution distorted fisheye video.

Chapter 7 summarizes the contributions of the thesis and discusses possible avenues for future research.

Chapter 2

Literature Survey

This chapter presents a comprehensive literature overview of state of the art algorithms for tracking people in 3D space. Also a review of 2D trackers specialized on tracking faces is given in the end of the chapter, in Section 2.2.

2.1 3D tracking

The main target of this research is whole body view 3D face tracking when a face typically occupies less than $5 - 10\%$ of a frame area. In this section, I discuss the most popular methods used for 3D tracking in sequences containing the whole body view of one or more people. I do not review here multiple works on 3D face tracking with a head and shoulders view. Such tracking methods are not directly relevant to my research because the face occupies most of the image and its 3D position is less important than its orientation and positions of facial features.

According to Wang *et al.* (2003) and Turaga *et al.* (2008), whole body view 3D people tracking include such application areas as surveillance, human-machine interaction (advanced user interfaces, gaming, virtual reality, etc), actions recognition and tracking for motion-based diagnosis, identification, or sports analysis and others. I do not review articulate motion tracking such as joints and limbs tracking as I am not interested in hands or legs positions. I am interested in 3D head or face tracking, but also consider whole body tracking as it can be an initial step to locate the face. Therefore in this review I concentrate on surveillance and human-machine interaction (HMI) applications.

Often the application for which the 3D tracking is performed defines the choice of sensors, their position and methods used for 3D tracking. Surveillance tasks include

people tracking in public areas and analysing specific behaviours in the sensed area (e.g. airports for malicious behaviour, hospitals for elderly or disabled people having issues) among others. The traditional approach for surveillance applications is to use a network of single cameras with no overlapping views. Such systems do not perform 3D tracking and are not reviewed here. In recent years, more authors suggested to use a network of stereo cameras instead of single cameras (e.g. Darrell *et al.* 2001, Harville 2004, Tao Zhao *et al.* 2005). Most often in such surveillance systems, the direct 3D position of people is not important, but the stereo information is used to supplement luminance information for easier foreground segmentation and tracking.

In contrast human-machine interaction applications often employ the 3D position of the tracked person directly to instruct the interactive computer or robotic system for the following actions (e.g. in head coupled displays described in the works of Brar *et al.* 2010, Dundas and Wagner 2014, Johnsen *et al.* 2013, Lee *et al.* 2013). HMI systems include augmented and virtual reality, robotics, intelligent living environments and other applications.

2.1.1 Cameras number and position

For 3D tracking one, two or multiple cameras can be used. Using two cameras is the most common approach and most of the works described further use two cameras. It is the minimal stereo system required to make direct estimation of the target 3D position by triangulation. Usually in the two cameras stereo system, the baseline and the angle between the optical axes are small. This way the two cameras overlook the scene from two similar perspectives and it makes it easier to find accurate correspondences and estimate disparities (Llewellyn *et al.*, 2010).

In some works, several overlapping cameras with wide baselines are employed to obtain a better view of the scene. Khan and Shah (2006) use 4 cameras, while Mittal and Davis (2002) use 6 cameras overlooking the same scene from different perspectives. The baselines and the angles between the optical axes of each camera are large enough to view the same scene from completely different perspectives (see Figure 2.1 for an example). Such setup allows to deal with occlusions more efficiently, especially in crowded scenes. The downside of the wide baseline system is that it is very hard to match corresponding points in different views because they look so different. The disparity map that is often used for foreground segmentation cannot be computed in wide baseline systems.

It is possible to estimate a 3D position using just one camera. If the camera

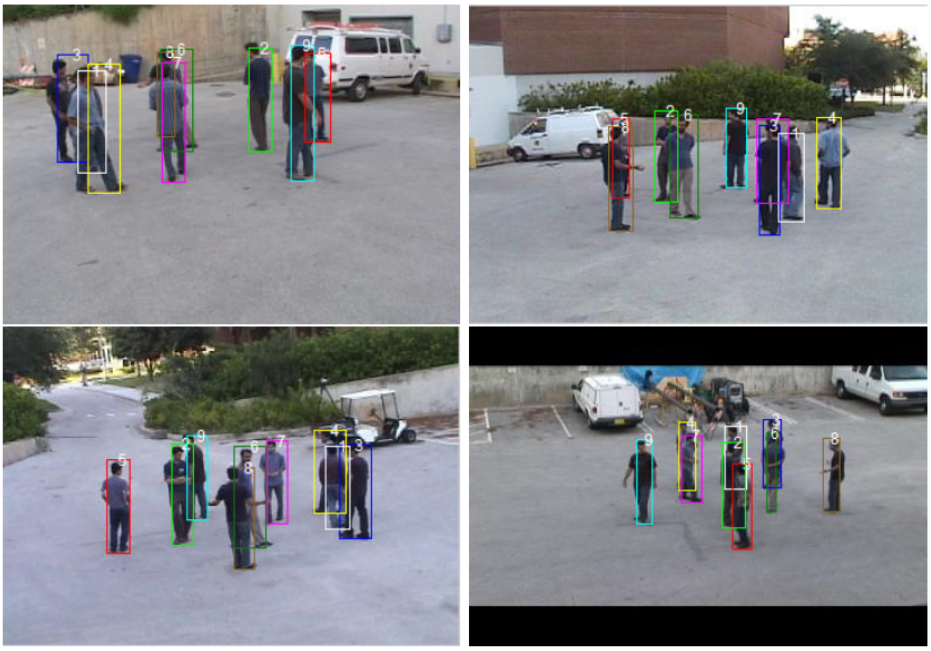

Figure 2.1: An example of a wide baseline system with four cameras. Four views from different perspectives allow to separate 9 people, even though they occlude each other in each view. The image and caption are taken from Khan and Shah (2006) with permission of Springer.

calibration is known, the target 3D position can be found by comparing its known real world size (e.g. the size of the human head or human height) to its projection size. Rougier and Meunier (2010a) use 3D particle filter to track the 3D head position by one camera. In this work the 3D head ellipsoid is projected into the 2D camera view. The tracked person's height is used to estimate the initial 3D position of their head.

Another possibility is to use structured light or time-of-flight (ToF) depth sensors. A ToF depth sensor emits modulated light to target objects and measures the phase delay of reflected light waves at each sensor pixel to calculate the distance travelled (Gokturk *et al.*, 2004). Structured light depth sensors project a pattern onto target objects, which provides a unique illumination code for each surface point observed by a calibrated imaging sensor (Maimone and Fuchs, 2012). Once the correspondence between the projector and the sensor is identified by stereo matching methods, the 3D position of each surface point can be calculated by triangulation. Kinect is the most popular and accessible depth sensor (Han *et al.*, 2013). Kinect V1 uses a structured light sensor. Kinect V2 uses a time-of-flight sensor. An overview of tracking algorithms using Kinect can be found in Han *et al.* (2013). Rougier *et al.* (2011), Mastorakis

and Makris (2014), and Stone and Skubic (2014) explore the Kinect sensor for the application of detecting falls of the elderly. The authors use a depth map obtained from Kinect to segment a human figure from the background and estimate the motion of its centroid. Luber *et al.* (2011) use three Kinects to get a wider field of view for tracking people. In their system, a local depth-change detector employing histogram of oriented depths (HOD) is formed, which is conceptually similar to histogram of oriented gradients (HOG) in RGB data (Dalal and Triggs, 2005). On top of that, a probabilistic model combining HOD and HOG detects the people from the RDB-D data. Choi *et al.* (2011) adopt five different observations to track humans in the image: motion, upper body appearance, face appearance, head and shoulders shape, skin color. The depth map is used to compare the candidate to the depth based shape template (see Figure 2.2). All these features are combined in a Bayesian framework. Tulyakov *et al.* (2015) use Kinect together with a luminance camera to accurately infer head pose, perform face frontalization and estimate facial expressions in real-time. Their system is designed to cope with a wide range of head pose variations, but Kinect and the camera need to have a close head-and-shoulder view of the tracked face. In general Kinect and other depth sensors give a fast way to get a depth map synchronised with a luminance image. In all other aspects, after producing the depth map, the tracking algorithms with Kinect are similar to the algorithms using stereo cameras. The downside of Kinect is a limited viewing zone (0.5 to 3.5 meters in depth) and viewing angle (57° × 70°).

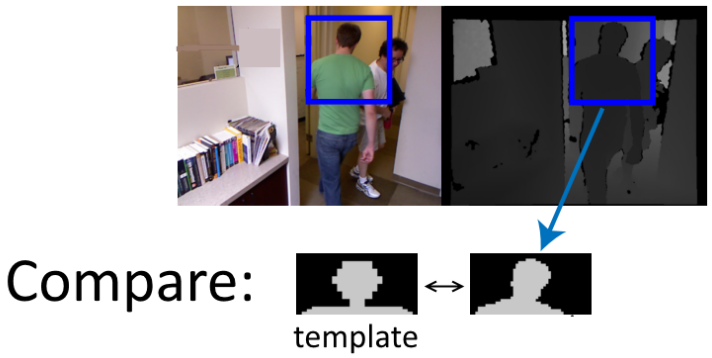

Figure 2.2: A filtered image of the tracked person's head and shoulders extracted from the depth map is compared to the head and shoulders template. This image is taken from Choi *et al.* (2011) ©IEEE.

Often surveillance applications differ from HMI applications in the positioning of cameras. Three possible positions are shown in Figure 2.3. Most surveillance applica-

tions use the top corner camera position (e.g Cai *et al.* 2010, Xiaoyu Huang *et al.* 2004, Bahadori *et al.* 2007). This way it is easier to place cameras in crowded public spaces and it helps to reduce occlusion. One of the drawbacks of this setup is that the face and the human body silhouette are not always clearly visible. Englebienne *et al.* (2009) use ceiling-mounted, straight-down cameras to count people in a supermarket. The advantage of this camera placement is complete elimination of occlusion in the central part of the view, but the field of view is very limited. Most HMI applications that perform tracking with luminance and/or depth cameras, use forward facing cameras when the camera is placed at near head height (e.g. Muñoz-Salinas *et al.* 2007, Darrell *et al.* 2000, Nanda and Fujimura 2004). This way the face is clearly visible when the user is facing the cameras. Also the hands are visible too and that allows to analyse gestures if needed.

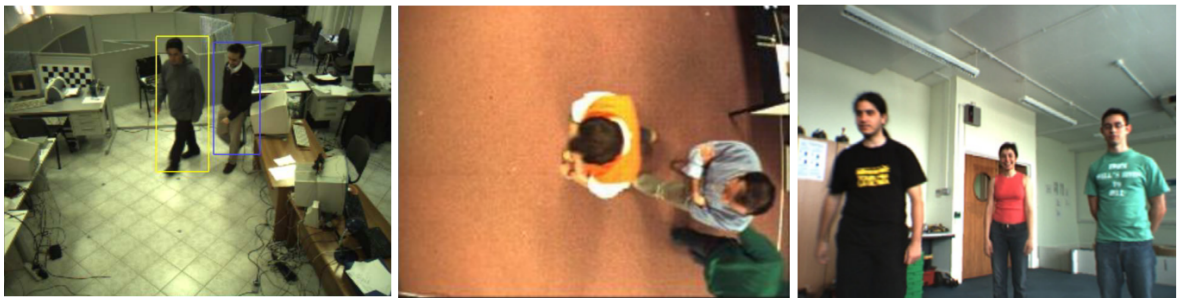

Figure 2.3: The top corner position is taken from Bahadori *et al.* (2007) with permission of Springer. The top down position is taken from Englebienne *et al.* (2009) ©IEEE. The front facing position is taken from Muñoz-Salinas *et al.* (2007) with permission from Elsevier.

2.1.2 Using stereo information

HMI systems and surveillance systems with stereo cameras use stereo information to extract the three dimensional structure of a scene. Extracting the 3D structure of the scene from stereo views is usually referred to as computational stereo in the literature. The fundamental basis for computational stereo is the fact that a single three dimensional physical location projects to a unique pair of image locations in two observing cameras. If it is possible to find the image locations that correspond to the same physical point in space, then it is possible to determine its three dimensional location.

The stereo computation field is usually divided into three problems:

1. Calibration

2. Correspondence

3. Reconstruction

Calibration: It is the process of determining camera system internal geometry (focal lengths, optical centers, and lens distortions) and external geometry (the relative positions and orientations of each camera). Accurate estimation of this geometry is necessary for relating image information (expressed in pixels) to an external world coordinate system. There are high quality tools available for estimating calibration, e.g. Camera Calibration Toolbox for Matlab[1], calibration tools in the OpenCV library[2], the OCamCalib tool[3] for wide angle cameras (Scaramuzza *et al.*, 2006).

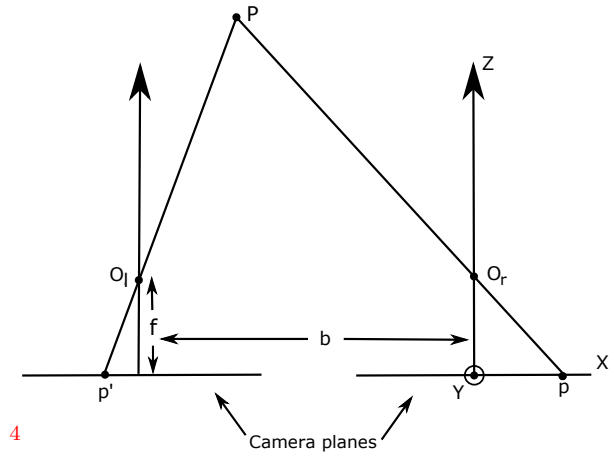

Figure 2.4: Schematic representation of stereo cameras setup.

Correspondence: The minimal possible stereo system is composed of a pair of cameras with optical centres O_l and O_r separated by a distance of b (see Figure 2.4). Let us assume that both cameras coordinate systems are axis-aligned and the baseline is parallel to the X coordinate axis. A point P in space projects to two locations on the same scan line in the left and right camera images. The resulting displacement of a projected point in one image with respect to the other is called **disparity**. The set of all disparities between two images is called a **disparity map**. Disparities can only be computed for the points that are captured in both images, but even this is not always possible because of occlusions and/or lack of texture.

[1] https://au.mathworks.com/help/vision/camera-calibration.html
[2] https://docs.opencv.org/2.4/modules/calib3d/doc/calib3d.html
[3] https://sites.google.com/site/scarabotix/ocamcalib-toolbox

The correspondence problem comprises of determining the locations in each camera image that are the projections of the same physical point in space. There is no general solution to the correspondence problem because of ambiguous matches (e.g. due to occlusion, specularities, or lack of texture). Thus, a variety of constraints (e.g. the epipolar constraint) and assumptions (e.g. image brightness constancy and surface smoothness) are commonly used to make the problem tractable. Correspondence methods include block matching, gradient based optimisation, feature matching, dynamic programming and other computer vision algorithms. The most recent and comprehensive review of all different methods of disparity map computation can be found in Tippetts *et al.* (2016).

Reconstruction: The reconstruction problem consists of determining three dimensional structure from a disparity map, based on the known camera geometry. The depth of a point in space P imaged by two cameras with optical centers O_l and O_r is defined by intersecting the rays from the optical centers through their respective images p and p' (Figure 2.3). Given the baseline distance b and the focal length f of the cameras, depth at a given point may be computed by similar triangles as

$$Z = f\frac{b}{d}, \tag{2.1}$$

where d is the disparity of that point converted to metric units. This process is called triangulation. To compute the depth map, a depth is calculated for each value in the disparity map.

The number of pixels that each image contains increases the number of calculations required to match it with any number of possible matches, making the correspondence problem a computationally complex one that severely limits the speed at which one can obtain results. However, stereo algorithms are easily parallelizable and therefore are well suited for implementation on graphics processing units (GPU), digital signal processors (DSP) or field programmable gate arrays (FPGA). Samarawickrama (2010) offers a detailed discussion about the advantages and disadvantages of these technologies with respect to real-time implementations of vision algorithms. Published performance results of stereo vision algorithms on GPU, FPGA and DSP-based platforms are provided in Tippetts *et al.* (2016) in an effort to understand the performance increases that are available when algorithms are optimized for parallel implementations on such hardware.

2.1.3 Foreground segmentation

The most common approach of employing disparity information is for extracting foreground from the disparity map. Eveland *et al.* (1998) and Darrell *et al.* (2000) segment foreground directly from a disparity map by thresholding and grouping. The process is illustrated in Figure 2.5. However, the accuracy of the foreground segmentation is affected by missing information such as that caused by occlusions, slanted surfaces, and other issues relating to extracting information about three dimensions from two dimensional images. Using the disparity map together with the luminance image for foreground segmentation helps to deal with this problem. One of the possibilities is to create a background model in luminance space and use it for the foreground segmentation in those parts of the image where disparity information cannot be retrieved. Different ways of combining disparity and luminance information for extracting foreground are used in Bahadori *et al.* (2007), Butt and Morris (2011), Harville (2004), Kalarot *et al.* (2012). Instead of using luminance and disparity information together, Tang *et al.* (2008) segment the foreground using just the luminance information and then back-project it to the depth map to get a point cloud of foreground objects in 3D space.

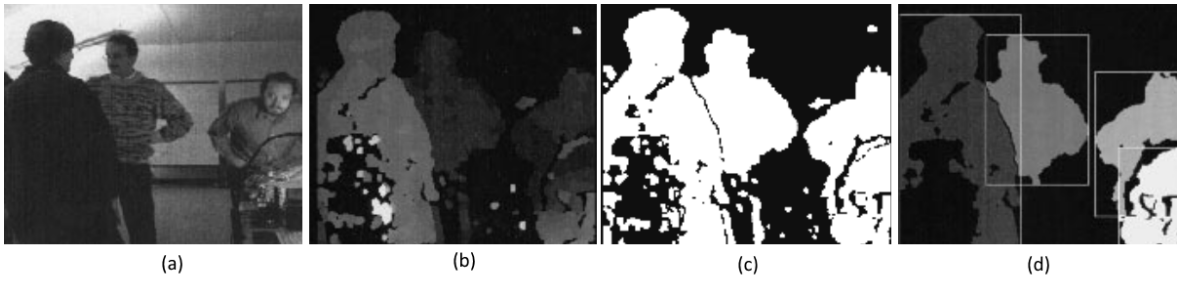

Figure 2.5: The foreground segmentation from a disparity map. (a) One of the two luminance images, (b) raw disparity map after morphological smoothing, (c) regions of slowly varying disparity, and (d) silhouettes recovered after connected components grouping. The image and caption are taken from Darrell *et al.* (2000) with permission of Springer.

Compared to simple depth map thresholding, the more accurate way of foreground segmentation is to use several depth maps computed at different moments to create a background model. When building the background model from several frames, a

mixture of Gaussians in the space of depth and luminance can be used to model each pixel. Then to filter out background, each pixel whose color and depth data match that pixel's background model, is labelled as background. The background model can be made offline, when there is no foreground present (Butt and Morris 2011, Kalarot *et al.* 2012). The downside of this approach is that the background can change, but the model is not updated. Another approach is to update the background model in real time. In the works of Darrell *et al.* (2001), Muñoz-Salinas (2007), Bahadori *et al.* (2007), Harville (2004), Tao Zhao *et al.* (2005) the background model is acquired online without a training sequence of foreground-free observations. Gathering background observations over long-term sequences has the advantage that lighting variation can be included in the background training set. Extreme lighting variation is useful, since it can cause previously uniform regions to have apparent contrast. Either the overall (ambient) or relative (shadow projected texture) illumination level can cause contrast to appear where previously the image region was uniform (and invisible in the depth map).

Rather then using depth map and/or luminance image background models, Muñoz-Salinas (2007) uses the height map background model created and updated in real time. A cloud of 3D points is calculated from the disparity map and projected into the floor plane. The floor plane of the observed scene is divided into bins and then the height of points that are projected to each bin are calculated. The process of creating the height map background model is illustrated in Figure 2.6.

2.1.4 Tracking Space

After the foreground is segmented, tracking is performed. Stereo-based object tracking is usually done in one of the three different spaces: camera-view space, plan-view space (also referred to as ground view or bird-eye view) or 3D space.

Tracking in camera-view space: Tracking in camera-view space means that the tracked object position is expressed in the pixel coordinates of one of the cameras' images. This camera is usually called the reference camera. Tracking in the camera-view space is often performed in HMI applications because in these applications the tracked person is usually close to the cameras, is in the foreground, occupies larger part of the frame and often there are no occlusions (e.g. Darrell *et al.* 2000, Nanda and Fujimura 2004, Argyros and Lourakis 2004).

The work of Darrell *et al.* (2000) is one of the early 3D HMI applications that track

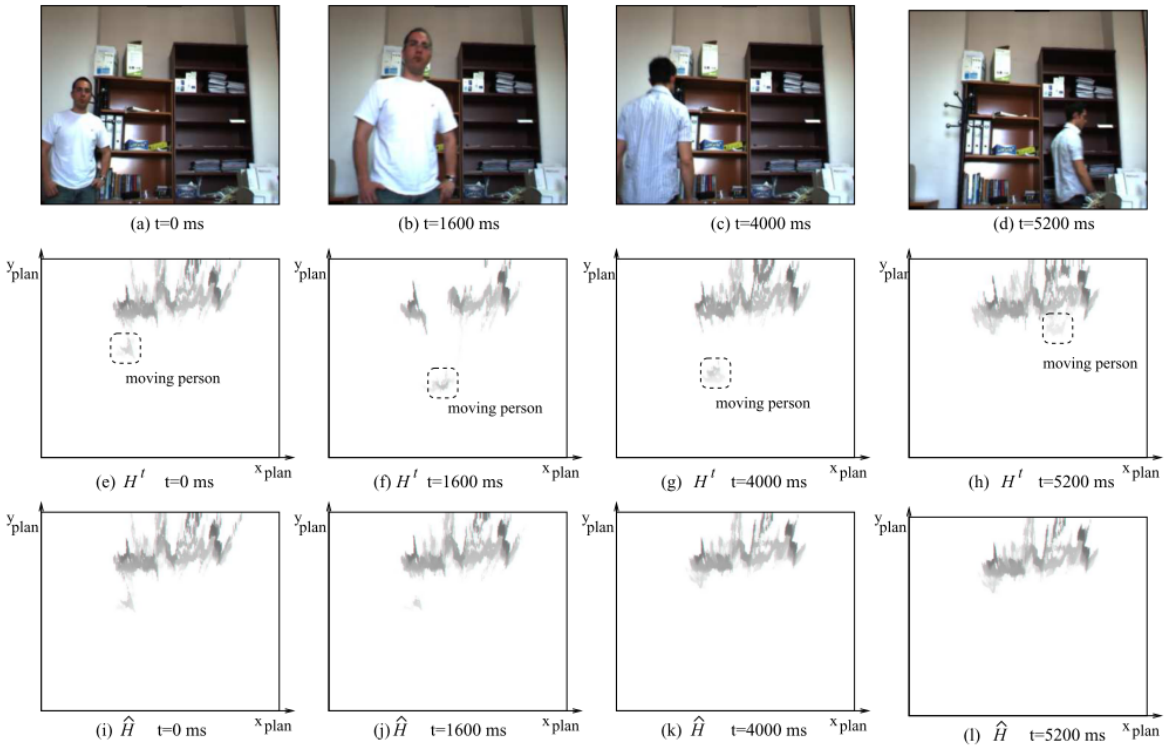

Figure 2.6: Segmenting the height map background model. The upper row shows frames in different time instances. The middle row depicts height maps in the same time instances. In the lower row the median of the height maps for each time instance is shown. In this row it is visible, how the foreground is gradually filtered out. The image is taken from Muñoz-Salinas (2007) with permission from Elsevier.

users' faces in the reference camera-view space. The depth map is used to segment silhouettes in the foreground. And then the skin color and face detector are used to localize faces in the reference frame. Another typical example of tracking in the camera-view space can be found in the work of Nanda and Fujimura (2004). They use the time-of-flight depth sensor and head and hands tracking is performed in the space of the sensor depth image. The edge map based on the depth map is employed to create an attractor basin, in which they try to fit body parts as shown in Figure 2.7.

In the work of Zoidi *et al.* (2014), the tracking is also performed in the camera-view space with the help of the Kalman filter framework. In this work, the color and disparity correlation is fully exploited by their combination in a 2-dimensional colour-disparity histogram. Each candidate object ROI is split into its three RGB color channels and for each channel the 2D colour-disparity histogram is computed. The histograms are

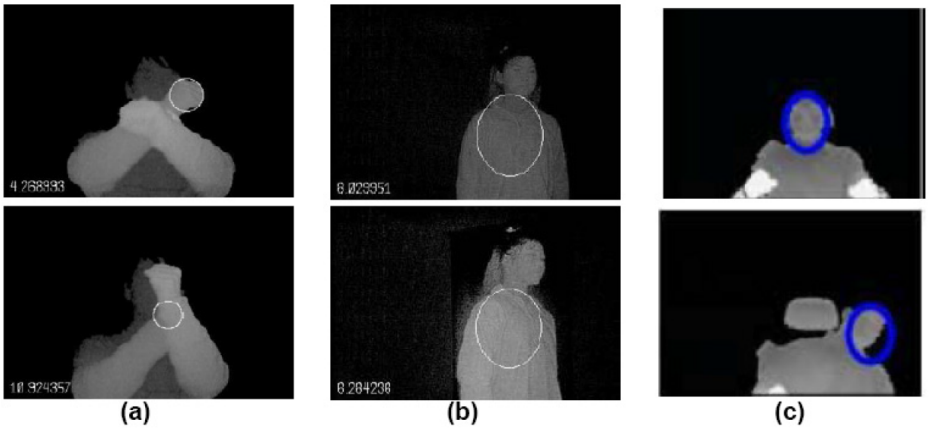

Figure 2.7: Tracking body parts in the depth map space (Nanda and Fujimura, 2004) ©IEEE. Tracking (a) hand, (b) body torso and (c) head.

constructed by selecting n_c bins for the color and n_d bins for the disparity information. See Figure 2.8 for an example of the three histograms. Then the colour-disparity histograms for each candidate ROI are compared to the histograms of the object detected in the previous frame. The ones with the lowest 2D colour-disparity histogram similarity are discarded.

Tracking in plan-view space: Tracking in plan-view space means that all detected features are projected onto the horizontal plane. Usually this plane coincides with the ground or floor. The tracked target position is expressed in a 2D coordinate system in the horizontal plane. Tracking in plan-view space is popular in surveillance applications (e.g. Harville and Dalong Li 2004, Bahadori *et al.* 2007, Khan and Shah 2006), but is also used in some HMI applications (e.g. Muñoz-Salinas 2008). The main advantage of this method is the total elimination of occlusion when tracking a crowd of people because two people cannot stand at the same spot on the ground at the same time.

Because the 3D point cloud is projected into 2D space, some information reduction happens. There are several ways to retain some of the 3D information when projecting it into the 2D plane. A common approach is to use plan-view statistics: height map and/or occupancy map. A conceptual flow diagram of plan-view statistics is shown in Figure 2.9. To compute plan-view maps, all reliable depth image values are back-projected, using camera calibration data and perspective projection, into a 3D point

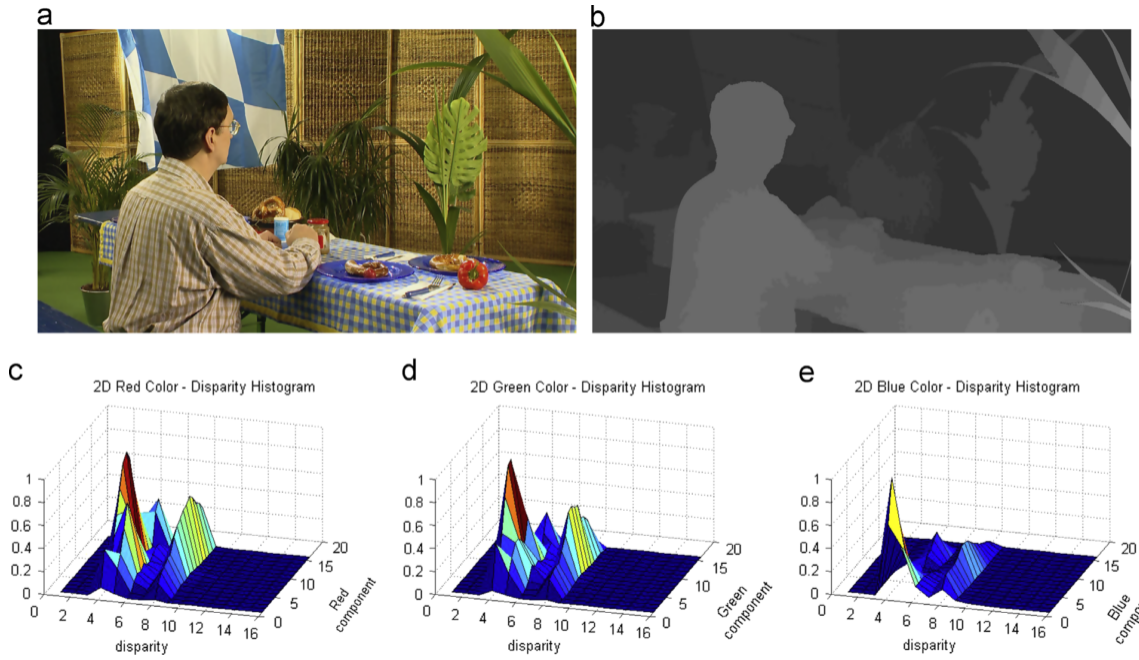

Figure 2.8: (a) Luminance channel, (b) corresponding disparity map, (c) 2D red channel-disparity histogram, (d) 2D green channel-disparity histogram, (e) 2D blue channel-disparity histogram. Image and its caption are taken from Zoidi *et al.* (2014) with permission from Elsevier.

cloud. This back-projection may optionally be restricted to the scene foreground. Then the space is discretized into a regular grid of vertically oriented bins and then statistics of the 3D point cloud is computed within each bin. Each pixel in a plan-view statistic image corresponds to one vertical bin, with the value at the pixel being some statistic of the 3D points within the corresponding bin. The occupancy map reflects the number of points in each bin, and the height map reflects the height of the highest point within each bin. When the plan-view projection is restricted to the foreground in the scene, the plan-view occupancy map provides an estimate of the amount of foreground at each floor location, while the plan-view height map indicates the shape of the foreground objects as viewed from above.

Tang *et al.* (2008), Harville and Dalong Li (2004), Muscoloni and Mattoccia (2014) segment the foreground first and then compute both the height map and the occupancy map. These two plan-view statistics can be computed through one pass of the depth map. Then different filters are applied to the occupancy map to find blobs that are similar in shape and size to a human figure viewed from above. The height map is

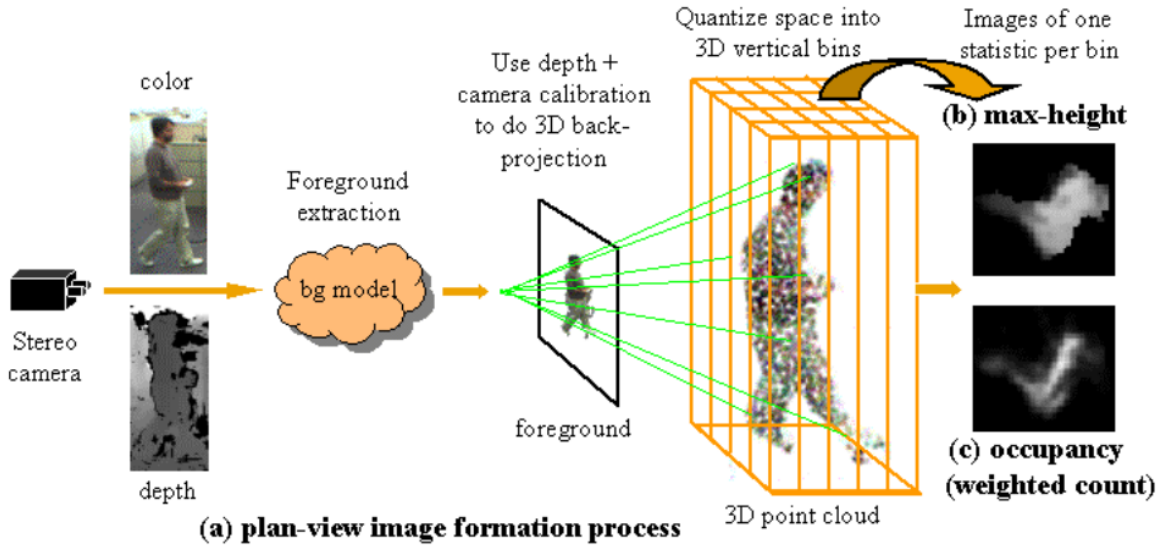

Figure 2.9: Generating a plan-view height map and occupancy map. Image is taken from Harville and Dalong Li (2004) ©IEEE.

used to filter out the blobs whose heights are not in the range of an average human height. Then an estimator such as a Kalman filter can be used to predict these blobs' positions either in the height map or in the occupancy map or in their combination in the next frame.

Besides the occupancy map and the height map, Muñoz-Salinas (2008) employed another plan-view statistic which is the colour map (see Figure 2.10). It contains the color histogram statistic of the 3D point cloud at each plan-view cell. The occupancy map and height map are used to detect objects, and colour map is used to create color models for the detected objects. A particle filtering-based tracking framework is used to track objects in plan-view space with the location, speed, and color information.

Boschini *et al.* (2016) employ a Convolutional Neural Network (CNN) to segment people using plan-view statistics. A colour image is submitted to the CNN trained to find heads. The CNN yields the positions of the heads' centres. The corresponding points on the disparity map are then projected in the plan-view perspective and a circle of radius δ is drawn around each of them thus obtaining a binary mask that will be used to filter out every point that is too distant from the detected head for it to belong to a person.

Bahadori *et al.* (2007) use just the height map for the plan-view statistics of the foreground. For each foreground blob in the height map, they compute two measures:

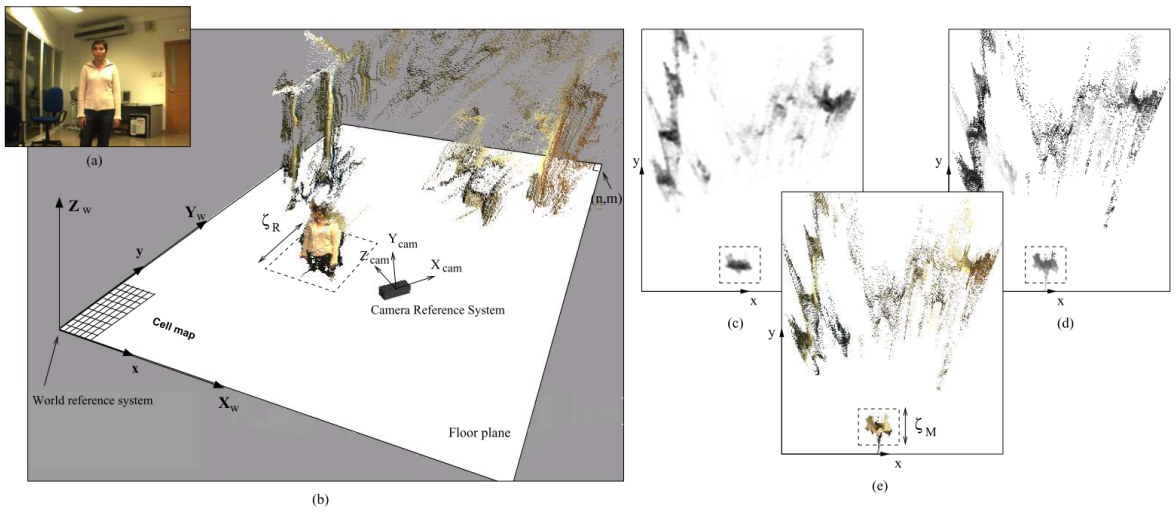

Figure 2.10: (a)Camera image. (b) 3D reconstruction of the scene. (c) Occupancy map. (d) Height map. (e) Coloured height map. The image is taken from Muñoz-Salinas (2008) with permission from Elsevier.

the size that is defined as the area of the blob in the X,Y plane and the weight that is the sum of the heights in the blob. The blobs are then filtered by removing those that have size or weight below given thresholds. This way the blobs corresponding to people are detected.

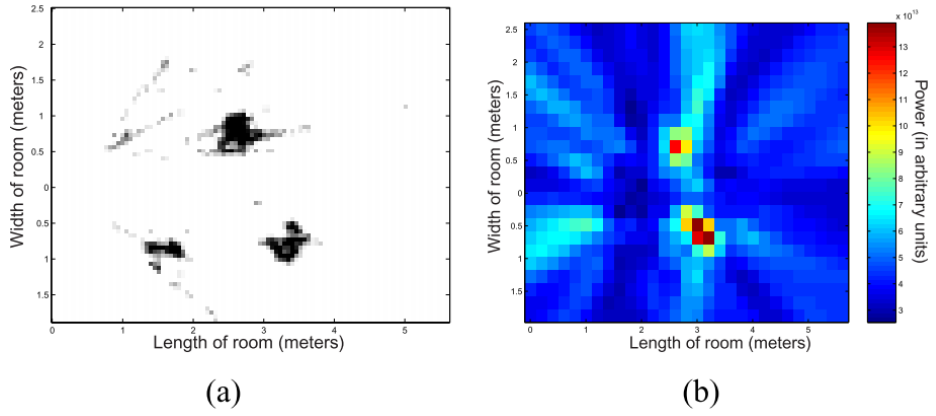

Figure 2.11: Observations consist of two maps. (a) A simpl binary plan-view image of foreground points. (b) A map of audio power in the microphone array coordinate system. Image is taken from Checka *et al.* (2003) ©IEEE.

Checka *et al.* (2003) compute a simple binary plan-view image of the foreground points. This binary map is used together with the 2D map of audio power obtained from an array of microphones (see Figure 2.11). A particle filter is used to combine observations from these two maps for tracking people.

Tracking in 3D space: When tracking in 3D space, the tracked target state is expressed as its 3D position. In the case of head tracking, this allows not to worry about the tracked target size. The real head size does not change, and the 2D size of the head projection into the camera image depends on how far is the head from the camera. Tracking in 3D space is often performed in HMI applications because the 3D position of the head is used directly for manipulating the machine and no extra steps, such as triangulation or 3D reconstruction, have to be performed. Another advantage of tracking in 3D space is that a tracking algorithm needs to evaluate only those pairs of image points in stereo views that make up a valid 3D location.

Nickel and Gehrig (2005) use a 3D particle filter to track the lecturer's head position. Each particle represents a 3D head position in the space of a classroom. Each particle is scored using features from both audio and video. On the video side, the features are based on foreground segmentation, multi-view face detection and upper body detection. On the audio side, the time delays of arrival between pairs of microphones are estimated with a generalized cross correlation function.

Butt and Morris (2011) use a Kalman filter for 3D tracking of pedestrians, where the object state is represented as the 3D position of a human body centroid and its 3D velocity. Further, by translating the human figure outline at predicted position in the real-world, a predicted disparity map by perspective projection of translated object outline is generated. The object in the current depth map is matched against the predicted shape at that location.

Kobayashi *et al.* (2006) perform 3D head tracking in a room with several cameras. A 3D particle filter is used for tracking. Each particle is a possible 3D position of the tracked head and its orientation. A likelihood of a human head is evaluated by several cascaded classifiers trained to detect a face in different orientations.

Rougier and Meunier (2010a) also use the 3D particle filter to perform 3D head tracking. In their work, the state of the head is represented by its 3D position and two rotation angles. The foreground segmentation, the body position, and head candidate color observations are used to estimate the likelihood of each particle.

2.1.5 3D tracking without building a depth map

As explained earlier, the depth map can be very useful for foreground segmentation or for computing plan-view statistics. However, the accuracy of results is affected by missing information such as that caused by occlusions, slanted surfaces, textureless regions and non-Lambertian surfaces along with the difficulties of perfectly calibrating cameras, and other issues relating to extracting information about three dimensions from two dimensional images. Also depth map computation is a very time consuming process, especially for high resolution images. The number of pixels that each image contains increases the number of calculations required to match it with any number of possible matches, making the correspondence problem a computationally complex one that severely limits the speed at which one can obtain results. A comparison of performance times for different stereo matching algorithms can be found in the paper of Tippetts *et al.* (2016).

Depth map computation is more difficult in stereo camera systems with a wide baseline between the cameras. A wide baseline generates a large viewing angle difference between an image pair which leads to more occlusions and more difficulties in stereo matching (Llewellyn *et al.*, 2010). Stereo systems with a wide baseline can be preferable in some applications because they allow higher depth accuracy when reconstructing 3D position of the tracked object (Chang and Chatterjee, 1992). There is no specific length that is considered to be a wide baseline. Usually a stereo system is considered to have a wide baseline if the cameras observe the same scene from strongly different angles (Llewellyn *et al.*, 2010). For example, cameras installed in the corners of a room and directed into its center have a wide baseline. In this situation, these two cameras have completely different views of the scene and it is very hard to find matching points because they look different in the stereo views.

Because of these problems, there are 3D tracking methods that use overlapping cameras and do not compute a depth map. Cai *et al.* (2010) and Abbaspour *et al.* (2014) detect a sparse set of features (corners and edge points). In the beginning, a static region of interest is defined manually to omit the background points which lie out of region (see Figure 2.12). The feature points in the region of interest are filtered by height and then projected on the ground plane. In the ground plane, clustering is performed to find people. The found clusters are tracked with the help of the Kalman filter.

Mittal and Davis (2003) use 6 cameras with the overlapping field of view to track a crowd of people. Instead of finding and matching feature points in different views,

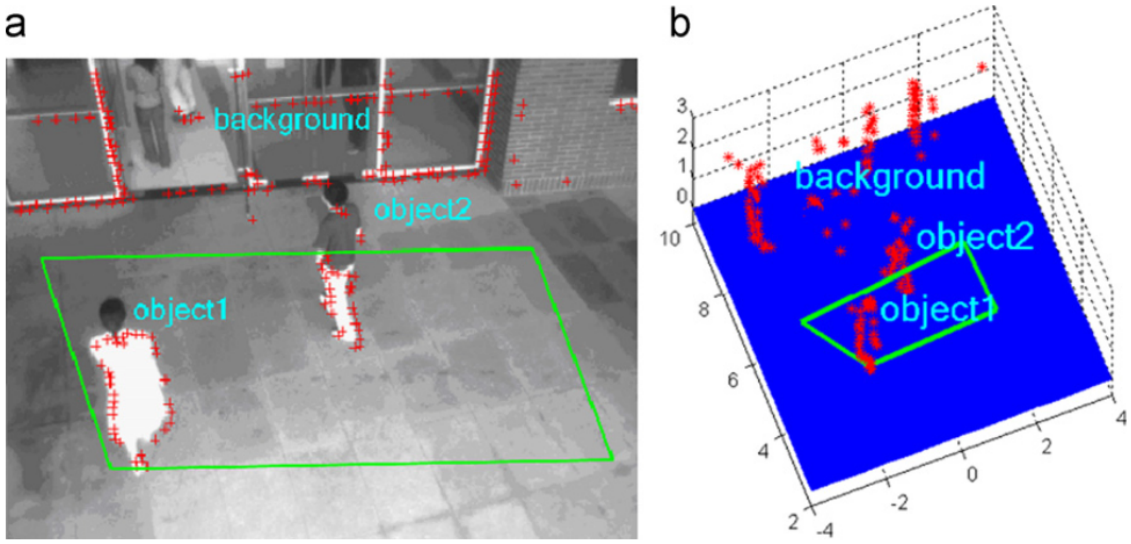

Figure 2.12: (a) Feature points extracted from the original grey scale image and (b) their projections to the world coordinate system. The green polygon is the region of interest used to remove background points. The image is taken from Cai *et al.* (2010) with permission from Elsevier.

region based stereo is used. They create a color model of each person at different heights using the method of non-parametric kernel density estimation (Elgammal *et al.* 2000). These color regions are matched in pairs of camera views along epipolar lines. The matched segments are then used to yield 3D points potentially lying inside objects (see Figure 2.13 for explanation). These points are projected on the ground plane and tracked with the help of the Kalman filter framework.

Stereo systems with a wide baseline often use a 3D particle filter for connecting information from different cameras and other sensors without building a depth map. Kobayashi *et al.* (2006) use a 3D particle filter together with several Haar cascaded classifiers to detect faces in different orientations. Nickel and Gehrig (2005) employ a 3D particle filter to track a lecturer's head position receiving observations from two wide baseline cameras and a pair of microphones.

2.1.6 Tracking methods

The three popular ways of tracking used in the 3D tracking applications involve mean shift, the Kalman filter and the particle filter frameworks.

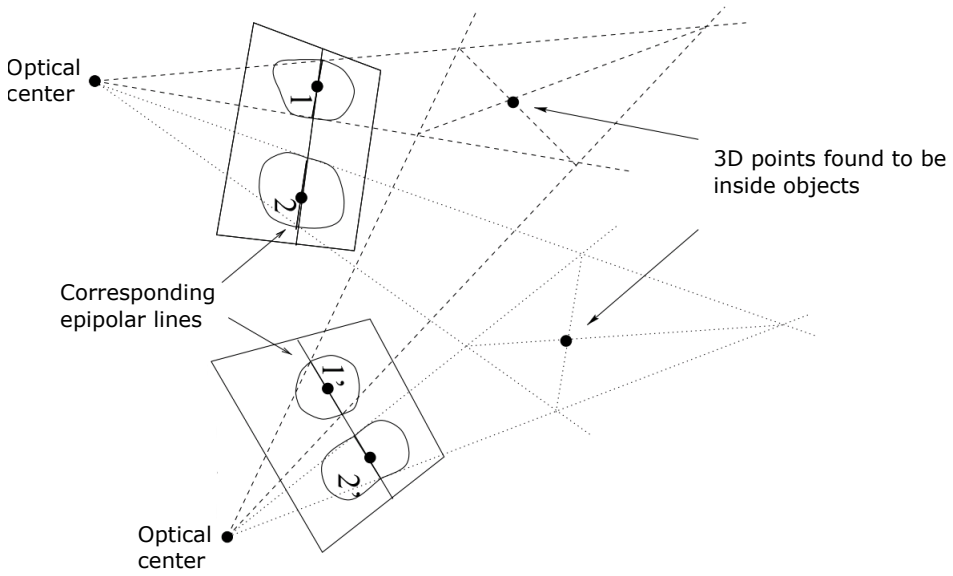

Figure 2.13: The concept of region based stereo matching. The point of intersection of the quadrilateral formed by back-projecting the end-points of the matched segments yields a 3D point lying inside an object. The matching segments are 1 and 1', and 2 and 2' respectively. This image and its caption are taken from Mittal and Davis (2003) with permission of Springer.

Mean shift is a kernel based gradient descent procedure that finds the local maxima/minima of a function (Cheng, 1995). When using the mean shift algorithm for visual tracking, a confidence map is created in the new image based on the object observations in the previous image. Then the algorithm finds the peak of this confidence map near the object's old position. The confidence map is a probability density function on the new image, assigning each pixel of the new image a probability, which is the probability of the pixel to belong to the tracked object.

Choi *et al.* (2006) adapt the color mean shift to use a disparity-weighted color histogram. They assume that the disparity distribution of the detected human body shows small variations with respect to the background regions or other objects. Kovacevic *et al.* (2011) implement a similar idea. They compute color probability distribution and disparity probability distribution of the region centred at the mean shift search window. Then they multiple and normalize these two distributions and get the joint

colour-disparity distribution and perform mean shift search on it. This helps to solve the problem with partial occlusion and when the background and the target have similar colors.

Mean shift based tracking algorithms are very efficient but they fail when facing occlusions or large target appearance changes or when the tracked object is under the action of non-linear forces. This method would not be appropriate in the case of tracking ambiguities that lead to running the optimisation procedure over a multimodal objective function.

The Kalman filter (Kalman, 1960) is another popular framework that is often used for tracking. It is a linear quadratic estimator that works in two steps. In the prediction step, it computes estimates of the current state variables and their uncertainties. Once the next measurement (corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The Kalman filter is often used to estimate a variable that cannot be measured directly, but an indirect measurement is available. Also it is often used to find the best estimate of the target state by combining measurements from various sensors in the presence of noise.

The Kalman filter is used in 3D tracking algorithms to predict the tracked object position taking into account the color and depth information (e.g. Abbaspour *et al.* 2014, Zoidi *et al.* 2014, Muscoloni and Mattoccia 2014, Cai *et al.* 2010, Harville 2004). Kalman filtering is often performed in the plan view space (Muñoz-Salinas *et al.* 2007, Mittal and Davis 2002, Kalarot *et al.* 2012, Bahadori *et al.* 2007), but can work in the 3D space as well (e.g. Butt and Morris 2011).

The Kalman filter is adequate for a simple linear model and runs faster due to computational simplicity. However, it does not perform well if used in a system that does not fit well into a linear model, or sensors' uncertainty is not Gaussian. In this case, the particle filter also known as the Condensation algorithm (Isard and Blake, 1998) and bootstrap filter (Gordon *et al.*, 1993) can work better by discretizing the problem into individual particles. Each particle is basically one possible state of the model, and a sufficiently large number of particles allows to handle any kind of probability distribution, and any kind of evidence. However, the amount of computations grows fast with the number of particles. Also a particle depletion problem can occur. Once the system runs out of particles in one area of the solution space, it's hard to get them back, so the correct estimate may just drop out permanently unless a very large number of particles is used.

Nickel and Gehrig (2005), Butt and Morris (2011), Kobayashi *et al.* (2006), Rougier and Meunier (2010a) use a 3D particle filter to estimate the target position in the 3D space. Muñoz-Salinas *et al.* (2008) use a particle filter to track the target in the reference camera 2D view. Checka *et al.* (2003) use combined probability for audio and video data in plan view space to track the target in the 2D particle filter framework.

2.1.7 3D tracking review summary

The choice of a particular hardware setup (number of cameras, their position, etc.) and methods used for 3D tracking is often defined by the final application. Surveillance and human-machine interaction (HMI) are the two most common applications of 3D people tracking. In this section, I summarize different hardware setup choices and methods and describe their advantages, disadvantages, possible applications and whether they can be used to achieve my research goal. I chose not to put citations here to make this review brief. All the appropriate citations are mentioned in the previous sections.

3D tracking system hardware setup choices:

- **Structured light or time-of-flight depth sensor versus standard cameras**. Kinect is a cheap and most popular depth sensor that is often used for HMI applications. Other industrial or custom made sensors are usually more expensive and/or not so easily available. Kinect gives a fast way to get a depth map synchronised with a luminance image. However the field of view of Kinect is limited and using multiple Kinect sensors poses additional challenges such as synchronization, interference and stereo calibration. Nevertheless Kinect achieves high 3D tracking accuracy and speed and can be used for whole body view face tracking in real time. I chose to use standard luminance cameras, to see if I can achieve similar robustness and accuracy.

- **Cameras number**. One, two or multiple cameras can be used for 3D tracking. Using one camera can be fast, but the accuracy of the 3D position estimation is limited. Multiple cameras are more often used in surveillance applications. Using multiple cameras with wide baselines gives a better view of a scene and helps to eliminate occlusion. However, processing frames from multiple cameras is time consuming. I chose to use a pair of cameras which is the most common way to do 3D tracking with luminance cameras in HMI applications. Using two cameras allows to achieve high 3D position estimation accuracy and speed of processing.

- **Baseline between the cameras**. Short baseline systems are most often used with a disparity map estimation which is a very time consuming process. Wide baseline systems allow higher depth accuracy when reconstructing 3D positions, but in such systems it is often impossible or very hard to compute a disparity map. In wide baseline cameras systems, usually a sparse set of features is used. I chose a medium baseline of 80cm with 30 degrees angle that gives reasonable depth resolution for the chosen operation range (see Section 3.2).

- **Cameras position**. In 3D tracking, one of the three cameras' positions is usually used: a top corner position, a ceiling-mounted, straight-down position or a front facing position. When using a top corner or ceiling-mounted, straight-down position, it is easier to eliminate occlusion and track several people at the same time. These cameras' positions are most often used in surveillance applications. A forward facing position is almost always used in HMI applications. When using this cameras' position, the tracked person's face is clearly visible. I do not need to track several people simultaneously and using face appearance can facilitate tracking. Therefore, I chose to use the forward facing position.

Like 2D tracking algorithms, 3D tracking algorithms based on stereo cameras also use the same tracking cues such as location, color, shape, and other observations to model the tracked target appearance. A comprehensive review of appearance models used in 2D tracking can be found in Li *et al.* (2013). The extra depth information from stereo cameras improve the quality of location information.

This list summarizes methods that are specific for 3D tracking algorithms:

- **Disparity map**. A disparity map is often used for foreground segmentation or for computing plan view statistics. Disparity maps are very popular in HMI applications with short baseline stereo systems. With a disparity map, the 3D location of each pixel is available. Combining this information with standard features used in 2D tracking such as color or shape can give good results. However, building a disparity map is a highly computational process and might take a lot of time, especially for high resolution video feed.

- **Feature points detection and clustering**. This method does not employ disparity maps and is often used in wide baseline stereo systems in surveillance applications. Such challenges as proper matching of points in different views and filtering out background points arise.

- **Plan view statistics**. This method involves projecting a depth map or a sparse 3D features set on the floor plane. Plan view statistics help to resolve occlusions and filter out background.

- **Deep learning**. Training a convolutional neural network on color images combined with disparity maps is a very promising method. There are not many papers on 3D tracking based on this method now (e.g.Boschini *et al.* (2016)). The downside is that it needs an extensive dataset and a lot of training time.

The methods listed above are all very promising and can be tried out for my research task. However, I chose a different way. Even though there is an emerging interest in 3D tracking and many new applications based on it appear regularly, this field is not so well studied and described in the literature as 2D tracking. A large variety of different learning methods, appearance models and tracking features that have been thoroughly evaluated in many different combinations in 2D tracking papers (Yang *et al.* 2011, Li *et al.* 2013), have not yet been tried out in 3D tracking. It is possible that incorporating some of the methods from the 2D tracking field will likely make the 3D tracking performance even better.

Another problem with 3D tracking methods is that it is hard to compare them quantitatively and find the one that suits better for a particular task. This can be explained by a large variate of setups that can be used for 3D tracking and initialization parameters that are needed to start the tracking. Therefore, it is hard to create a dataset that will suit a large number of methods to try them out. This is not the case for generic 2D tracking algorithms. They all run on image sequences and most of them need just a bounding box for initialization. There is a number of frameworks that put together state of the art 2D generic tracking algorithms and a number of evaluation datasets (e.g. Smeulders *et al.* 2014, Wu *et al.* 2015). This allows to compare quantitatively these algorithms and find the one that suits best of all for a specific goal.

For these reasons, I took an approach that is different to the classical way of developing 3D tracking applications. I decided to use the power of all latest improvements in the 2D tracking field and choose one of state of the art 2D tracking algorithms that shows best performance results on public datasets and on my datasets for whole body view face tracking. After finding such tracking algorithm I explored different ways to improve its accuracy even further by adapting it for tracking faces. At the final stage of my research, I explored different ways of extending this 2D face tracking algorithm into 3D space.

2.2 2D face tracking

2D visual tracking has always been a highly active research area due to a large number of potential applications. The interest in this area has led to a significant amount of literature. A review of advances and trends in visual tracking can be found in Yilmaz and Javed (2006), Cannons (2008), Yang *et al.* (2011). Li *et al.* (2013) made a comprehensive review of appearance models used in tracking. In recent years instead of just reviewing state of the art trackers, more authors try to perform a fair, extensive and unbiased evaluation of different trackers (e.g. Smeulders *et al.* (2014) and Wu *et al.* (2015)).

For a large number of tracking scenarios, the visual class of the object of interest is known. These tracking algorithms are different from generic trackers because they integrate some prior knowledge about the object class. The most straight forward way to do that is to train an object class detector in advance and integrate it into the tracking process. Face tracking is one example of such tracking scenarios.

A very popular method of face tracking is fitting deformable models of facial landmarks (e.g. Saragih *et al.* (2011), Asthana *et al.* (2014)). A parametrized model of facial landmark locations is created. The image analysis is performed by fitting the deformable model to a new image, thereby parametrizing the new image in terms of the known model. A simple tracking algorithm such as mean shift and different feature detectors can be used to predict new locations of landmarks. These specialized face tracking methods show great results for tracking faces and facial features. They are less likely to drift away to the background. However all these methods are designed to work on faces with facial landmarks that are clearly visible. This is often the case for the head-and-shoulders view videos, when a person sits in front of a camera. These methods can not be applied to the whole body view tracking scenarios, because the tracked face is often too small and facial landmarks are hardly distinguishable. Pham and Pavlovic (2016) use Kinect in addition to a luminance camera to fit a 3D blendshape face model to the face region in color and depth streams. Their method showed reasonable results when tracking faces at the distance up to 2.5 meters on synthetic datasets. In this research, my goal is to do face tracking at the range of 0.5 meters to 5 meters using just luminance cameras (see the requirements analysis in Section 1.2).

Recently, features extracted from a convolutional neural network (CNN) trained on a large-scale object recognition dataset have been successfully applied to face recognition, identification, and clustering tasks (e.g. DeepID by Sun *et al.* (2014), DeepFace

by Taigman *et al.* (2014), FaceNet by Schroff *et al.* (2015), VGG-Face by Parkhi *et al.* (2015)). These CNN-based face representations are learned by training CNNs using large-scale face recognition datasets in a fully supervised manner. Zhang *et al.* (2016) propose to use a pre-trained CNN to learn video-specific features in an unsupervised manner and then track multiple persons through an unconstrained video sequence such as a sitcom episode. This method is very promising, but the downside that it is designed for offline processing. Several runs through the same sequence a needed to adapt the pre-trained CNN and then correct the initial characters' trajectories.

Whole body view face tracking is often used in surveillance. Face tracking algorithms designed for surveillance usually use a wide-angle camera to detect and track faces or silhouettes and a pan-tilt-zoom camera to get a closer view of a face Bagdanov *et al.* (2006), Park *et al.* (2013). The main challenge faced by such systems arises in registering the pan and tilt angles of the PTZ camera and a complex calibration procedure between the two cameras. These algorithms mostly concentrate on getting high-resolution images of pedestrians' faces, but fail to do continuous smooth tracking with reacquisition.

Another class of whole body view face tracking algorithms utilise face specific information by integrating some sort of a face detector into the tracking process. Using a face detector can help to handle tracking with a large scale change (from the whole body view to the head-and-shoulders view) together with rapid movements and occlusions while distinguishing different people's faces. Li *et al.* (2008) proposed using three face observation models with different feature sets and different lifespans: short (updated every frame), middle (updated every five frames) and long (a face detector, learned offline) (see Figure 2.14 for illustration). These three models are combined in a cascade fashion in the particle filter framework.

Kim *et al.* (2008) use non-adaptive, non-person specific constraints on face pose and localization to adapt the incremental visual tracker (IVT, Ross *et al.* 2008) for tracking faces. These constraints are learned offline. They include a generative model and a discriminative model. The generative model contains a set of facial pose subspaces or manifolds, each of which represents a particular out-of-plane pose. The discriminative model is based on a support vector machine (SVM) that helps to estimate how well the cropped face is aligned (see Figure 2.15).

Both methods described above (Li *et al.* 2008 and Kim *et al.* 2008) are based on the particle filter framework and are able to track the target in the field of view. In scenarios where a face moves in and out of the viewing zone, these trackers usually fail

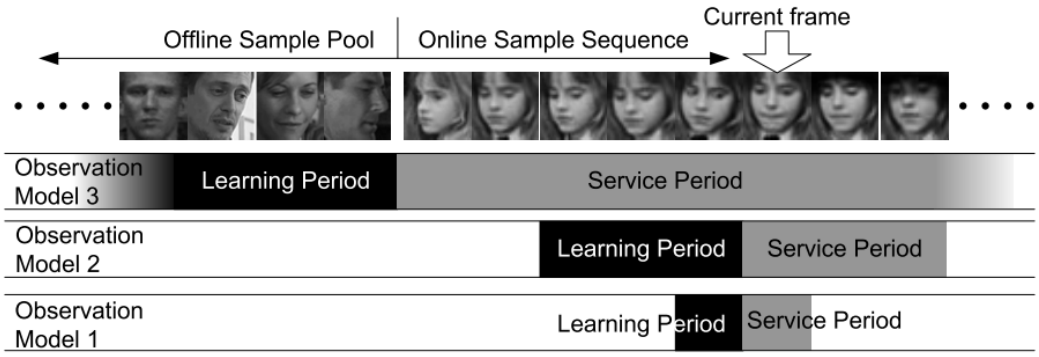

Figure 2.14: Using three models with different life spans for face tracking. Image is taken from Li *et al.* (2008) ©IEEE.

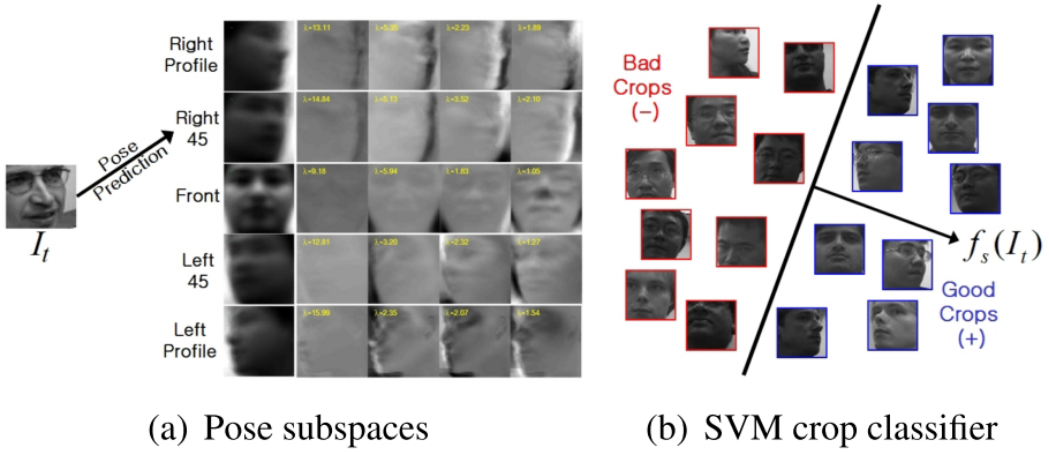

Figure 2.15: Visual constraints for face tracking used in the work of Kim *et al.* (2008). (a) A generative PCA model where each row represents a certain pose subspace. To predict a pose I_t, the shortest distance between the current tracked face I_t and all the subspaces is found. (b) SVM classifier that discriminates face-centred images (+, in blue) against badly cropped face images (−, in red). This image was taken from Kim *et al.* (2008) ©IEEE.

because they do not have a face reacquisition system.

FaceTLD by Kalal *et al.* (2010a) uses a combination of a simple optical flow tracker and an offline trained face detector. Structural constraints in video are used to build a model of the tracked face. The build face model is utilized to correct the tracker or reinitialize it if it loses the target. This method is discussed in details in Section 4.3.1.

In general using some sort of face detection in combination with a robust generic tracker can be very beneficial for whole body view face tracking. A face detector (or a combination of detectors) that can detect faces in different orientations is even better. Tracking body parts or whole body silhouette can also help. On the contrary tracking facial features such as eyes or mouth is not useful because in whole body view videos these features are often too small.

2.3 Conclusion

In this chapter I reviewed state of the art algorithms for whole body view people tracking in 3D space. Works on face tracking were also reviewed in this chapter. Based on this review, I decided to follow a non classical approach and develop a 2D face tracking algorithm first. Next chapter describes all preparation steps required for designing 2D and 3D tracking algorithms: stereo calibrating cameras, recording and annotating evaluation datasets, choosing metrics that will be used to compare different algorithms. The process of developing 2D face tracker for whole body view video is described in Chapter 4. Then in Chapter 5, I investigate different ways of converting the developed 2D face tracking algorithm into 3D space.

Chapter 3

Tracking evaluation framework

This chapter describes all preparation steps required for designing a tracking algorithm. For developing a successful tracker, it is necessary to compare it to other publicly available state-of-the-art trackers and to different versions of itself. The evaluation should be performed on a dataset with ground truth annotation. A fair and representative metric is needed for interpreting the results.

Section 3.1 describes how the evaluation datasets were collected. Section 3.2 focuses on the stereo cameras system used for recording the datasets and real time tracking evaluation. The datasets annotation process is described in Section 3.3. The evaluation metric is chosen in Section 3.4.

3.1 Collecting evaluation datasets

For comprehensive tracking performance evaluation, it is necessary to collect a representative testing dataset. Running trackers on such a dataset helps to fairly compare them and reveal their weaknesses. A representative dataset should contain different scenarios and challenging factors specifically for the chosen tracking task (e.g. pedestrian tracking, car tracking, tracking elderly people for fall detections, face detection and recognition in an airport gate). As the goal of this research is long term whole body view tracking, I need to tailor the evaluation dataset specifically for this task. The dataset should comply with the requirements listed in Section 1.2. Videos in my datasets should feature a person moving in front of the cameras and facing the cameras most of the time. I consider the user moving freely in front of the cameras, therefore it is the whole body view, not the head-and-shoulders view that can be achieved by recording a video on a hand-held smart phone or a desktop web camera. Furthermore,

it should be a stereo dataset made with two cameras with an overlapping field of view. Each feed separately can be used for testing long-term 2D face tracking. The other challenging factors are described in Table 3.1.

Recently, significant efforts have been made to collect large public annotated datasets with different challenging factors for testing single object 2D tracking. The most notable datasets are the object tracking benchmark by Wu *et al.* (2015) (100 sequences), the Amsterdam Library of Ordinary Videos for tracking by Smeulders *et al.* (2014) (315 sequences) and the VOT challenge 2015 dataset by Kristan *et al.* (2015) (356 sequences). These datasets include videos for evaluating 2D tracking of pedestrians, cars, sportsmen, animals and other objects.

The 3D tracking evaluation datasets can be divided into two main applications: surveillance and human motion recognition. One example of 3D tracking datasets for surveillance is the KITTI vision benchmark suite (Geiger *et al.*, 2012). It contains 50 annotated stereo videos of cars and pedestrians on crowded streets. The stereo calibration information is provided. It also contains videos for optical flow, visual odometry/SLAM and 3D object detection. The surveillance datasets are not suitable for my research because people in videos are usually far away from cameras and looking away from the cameras.

HumanEva (Sigal *et al.*, 2010) and i3DPost (Gkalelis *et al.*, 2009) are examples of articulated human motion and pose estimation datasets. Each sequence in these datasets is recorded using several overlapping calibrated cameras and features a person moving in a limited space. The ground truth data contains all body parts and joints annotations for each frame. These datasets can be used for evaluation of my tracking algorithm. However, as it is designed not for 3D face tracking, but for pose estimation, the ground truth data has to be customized. For each pair of cameras, I have to choose the frames when the face is visible and extract its 3D position from the provided ground truth data. Also I could not find any 3D head or face tracking algorithms reporting their results on these datasets. Therefore, I decided to record my own datasets. One of the contributions of this research are three new extensive annotated datasets created specifically for evaluating long term whole body view 3D face tracking. I collected the following datasets:

- **2D dataset with public videos**. I examined the object tracking benchmark (Wu *et al.*, 2013) and selected 17 videos that contain a person facing the camera. For some examples, see Figure 3.1. The resolution ranges from 128 × 96 to

[1]Short video example of the "Asteroid" game with Kinect https://youtu.be/61wpZArEKuc

Factors	Occur.	Explanation
Occlusion	Yes	The tracked person can hide his or her face with the hands, or another person can walk between the tracked person and the cameras.
Fast motion and motion blur	Yes	If the tracking algorithm is used in an head-coupled display, the tracked person can play some game on the display that includes rapid movements, e.g. evading flying objects [1].
Scale variation	Yes	The tracked person can go up to 4-5 meters away from the cameras and then come back close to the cameras.
Rotations	Yes	In-plane and out-of-plane head rotations are possible.
Out-of-view	Yes	The tracked person can leave the cameras' field of view and return some time later.
Background clutters	Yes	The background near the target can have color or texture similar to the target or there might be other people in the background.
Low resolution	Yes	Some of my evaluation videos have low resolution.
Camera moving	No	The cameras' position is fixed.
Illumination variation	Possible	I assume that the tracking is performed in an indoor environment, but lightning conditions can change (see Figure 1.5 for an example).
Deformation	No	The tracked person can move freely in front of the cameras. Most of the time the whole body of the person is visible in each frame and the head occupies a small portion of the image. In this case, the head can be considered as a rigid object. Therefore no deformation is present.

Table 3.1: The table describes common challenging factors and their occurrence in my evaluation datasets.

720×576. The videos are 300 to 1500 frames long. This dataset is used to compare different public 2D tracking algorithms.

- **My high resolution gray scale stereo dataset**. The dataset was collected using two stereo calibrated gray scale Ximea cameras with $79° \times 59°$ angle lenses at the rate of 30 frames per second and resolution of 1280×1024 pixels. For some examples, see Figure 3.2. The testing sequences are 600 to 3000 frames long featuring a person freely moving in front of the cameras. The sequences include all the challenging factors described in Table 3.1. The dataset consists of 23 pairs of videos and is used for evaluating different versions of my 3D face tracking algorithm. Also for evaluating 2D tracking, one feed (from the left or right camera) of each stereo sequence is used.

- **My low resolution gray scale stereo dataset**. This dataset was created by resampling my high resolution dataset at the resolution of 640×512. The motivation for creating this dataset is to check how much speed up can be achieved by trading off accuracy.

- **My fisheye color stereo dataset**. A separate stereo dataset was created using cameras with Omnitech Robotics ORIFL190-3 fisheye lenses with the field of view of 190°. The motivation for creating this dataset is to evaluate how the 3D face tracking algorithms perform in videos with a wide field of view and high distortion. For the task of 3D face tracking for interactive displays, having a large joint field of view of the two stereo cameras allows smooth and convincing experience when using the display. The dataset consists of 10 pairs of videos. Each video is around 2000 frames long and was made at the rate of 30 frames per second with the resolution 640×480 pixels. Similarly to my other datasets, this one features a person moving in front of the cameras and facing the cameras most of the time. The sequences include all the challenging factors described in Table 3.1. For some examples, see Figure 3.3. Also at the same time as recording this dataset, I recorded the Kinect face tracking data. This allows to compare the Kinect head tracking method with other visual trackers on fisheye video. You can read more about capturing the Kinect data in Section 6.1.

3.2 Stereo setup and calibration

In this section I describe the stereo system used for recoding the evaluation datasets and qualitative evaluation of tracking algorithms in real time. The stereo system consists of two synchronized cameras with an overlapping field of view. More cameras can be

Figure 3.1: Examples from the 2D dataset with public videos.

Figure 3.2: Examples from my high resolution gray scale stereo dataset.

Figure 3.3: Examples from my fisheye color stereo dataset.

used, but for many 3D face tracking applications, two cameras are enough to cover a large area in front of the cameras. For recording the high resolution gray scale dataset,

I used two gray scale Ximea cameras[1] with Goyo lenses[2]. The Ximea cameras were chosen because they are relatively cheap high quality cameras. The field of view of the lenses is $79° \times 59° \times 98°$ ($H \times V \times D$). These lenses can be considered transitional between normal lenses and wide-angle lenses. They provide quite large field of view, but the distortion is still reasonable.

The baseline and the angle between the cameras should be chosen carefully. The baseline affects the depth resolution that refers to the accuracy with which a stereovision system can estimate changes in the depth of a surface. In the stereo system where the cameras are axis aligned, the depth resolution δZ is computed by the following equation:

$$\delta Z = -\frac{Z^2}{fb}\delta d, \tag{3.1}$$

where f is the cameras focal length, b is the baseline distance and δd is the disparity resolution (Chang and Chatterjee, 1992). It means that a good depth resolution requires a large baseline value, a large focal length value, and a small depth value for a given disparity resolution. The baseline and the angle between the cameras were chosen for the depth range of 0.5 - 5 meters from the cameras (according to the requirements set in Section 1.2). The baseline of 80 cm is considered large enough to give a good depth resolution for the chosen depth range. The angle between the cameras is chosen to be 30° to move the stereo system viewing zone closer to the cameras. The shared viewing volume is shown in Figure 3.4.

To be able to compute a 3D position of an object present in the cameras shared field of view, the cameras have to be stereo calibrated. A stereo calibration involves computing each camera's intrinsic parameters (the camera matrix and distortion coefficients) and the extrinsic parameters (the camera coordinate system position and orientation in the world coordinate system). Usually in the stereo cameras setup, one camera coordinate system is assumed to coincide with the world coordinate system. In this case the extrinsic parameters are the rotation matrix and the translation vector needed to transform from one camera coordinate system to the other camera coordinate system. To stereo calibrate the cameras, I used a chequerboard pattern to find the corresponding points in the two cameras' views and the Matlab stereo camera calibration application which is based on the Zhang (2000) calibration algorithm. This tool can

[1]Model https://www.ximea.com/en/products/usb3-vision-cameras-xiq-line/mq013rg-e2

[2]Model http://www.rmaelectronics.com/goyo-optical-gmthr24514mcn-1-2-4-5mm-f1-4-manual-iris-c-mount-lens-3-megapixel-rated/

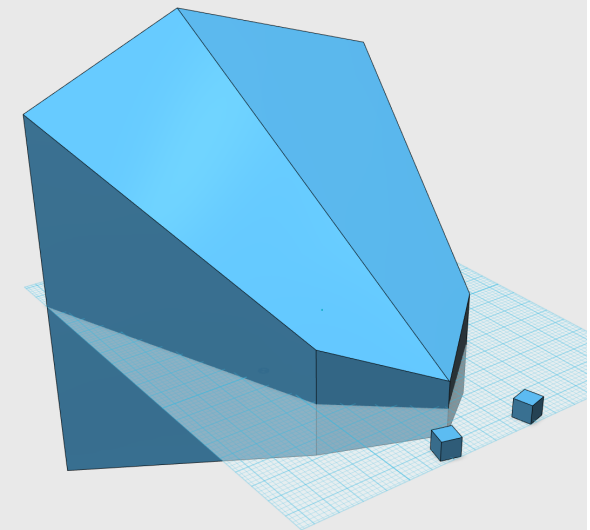

Figure 3.4: The shared viewing volume of the stereo system.

be used for cameras with the field of view up to 95°. The average reprojection error is normally employed to estimate the calibration accuracy. The reprojection error is a geometric error corresponding to the image distance between a projected point and a measured one. It is used to quantify how closely an estimate of a 3D point recreates the point's true projection. For the stereo calibration that I performed, the average reprojection error was 0.55 pixels. This error makes up 0.05% of the image height which can be considered a good result.

For recording the fisheye dataset, I used the Fire-i Unibrain cameras[1] with the Omnitech Robotics ORIFL190-3 fisheye lenses[2] with a field of view of 190°. The Matlab stereo calibration tool cannot be used for the cameras with such wide angle lenses. Instead I used the OCamCalib toolbox[3] (Scaramuzza *et al.*, 2006) for finding the intrinsic parameters of each camera. This toolbox is designed specifically for calibrating wide angle and omnidirectional cameras. In this calibration application the image projection function is described by a Taylor series expansion whose coefficients are estimated by solving a two-step least-squares linear minimization problem. For both of my fisheye cameras' calibrations, the resulting average reprojection error is around 0.7 pixels which is 0.15% of the image height.

After calibrating the two fisheye cameras, I performed stereo calibration to find one camera position and orientation in the other camera coordinate system. To do that

[1]Model http://www.unibrain.com/products/fire-i-board-camera/
[2]Model http://www.omnitech.com/fisheye.html
[3]https://sites.google.com/site/scarabotix/ocamcalib-toolbox

I used the OpenCV stereoCalibrate function that applies an iterative optimization algorithm to minimize the reprojection error of the chequerboard corners for both camera views. This function can perform each camera calibration for the intrinsic parameters internally or accept the camera matrix and the distortion parameters. As I calibrated the fisheye cameras separately by the OCamCalib tool, I provided the intrinsic parameters to stereoCalibrate. The baseline between the cameras is 105 cm and the angle between the optical axes is 0°. These baseline and angle were chosen to provide optimal viewing zone and depth resolution for the fisheye stereo cameras. The final average reprojection error is 0.8 pixels. This error is 0.17% of the image height which can be considered as an acceptable result.

3.3 Ground truth annotation

To be able to use the collected datasets for tracking evaluation, the ground truth annotation has to be obtained. I collected two large independent datasets: my high resolution gray scale stereo dataset with around 46000 frames in it and my fisheye color stereo dataset with 20000 frames. It is necessary to obtain the ground truth for every frame in these datasets. This task is very tedious and time consuming if not automated at least partially. In this section I describe two methods of fast ground truth labelling.

Figure 3.5: Different ground truth labelling formats: axis aligned bounding box, oriented bounding box, head contour.

Before choosing the ground truth labelling method, I need to select the format of the ground truth data. The most common ground truth formats for tracking algorithms are shown in Figure 3.5. The axis aligned box is the most rough and the easiest to

create format. The oriented box and the head contour are more accurate and take more time to generate. For my experiments I choose to use the axis aligned box. The center of the head bounding box usually coincides with the head center. Therefore, by triangulating the bounding boxes centers, it is possible to find the 3D position of the head. All current publicly available tracking algorithms discussed in this research, output axis aligned rectangular bounding boxes.

After choosing the ground truth format, it is important to find a way to do fast and accurate labelling. In recent years many different annotation tools have appeared for reducing the human effort necessary to generate ground truth data for large scale visual datasets and improve the annotation quality. Most of these tools allow the annotation of only a few frames (i.e. key frames) and then propagate the annotation by means of dedicated algorithms (Mihalcik and Doermann 2003, Yuen *et al.* 2009). These tools incorporate a form of basic annotation propagation exploiting the visual coherence of neighbour frames and more sophisticated computer vision and machine learning methods to automatically label some of the frames (Kavasidis *et al.* 2012, D'Orazio *et al.* 2009, Bianco *et al.* 2015). Some annotation tools promote the use of crowd-sourcing based platforms to improve the quality of the annotations (Kavasidis *et al.* 2014, Kavasidis *et al.* 2013, Vondrick *et al.* 2013).

All the tools described above can be really helpful when annotating large datasets. However, these tools are generic software solutions designed to create ground truth data for different computer vision datasets. It takes some time to choose one of them, install and understand the complex interface and customize it for specific purposes (head tracking in my case). Instead, I developed my own method of annotating datasets described further.

As mentioned above, many of the recent ground truth labelling tools use annotation propagation and machine learning methods to partially automate the annotation process. Most often a tracking algorithm or detection algorithm is used to propagate the target annotation marked manually by the user to the next frame. I decided to do ground truth labelling using this principle and chose the TLD tracking algorithm (Kalal, 2011) for this task. As discussed later in Section 4.1, TLD shows superior results over other public tracking algorithms and is also very fast. To use it for ground truth labelling, I initialize TLD manually on the first frame of a video and then allow it to do the tracking. I changed it slightly, so that it moves to each next frame after I press the ENTER button. That gives me time to quickly review that the tracking is still correct. If the tracker lost the target or the tracking box starts to diverge from

the target, I press the Q button to stop the tracking and reinitialize it. For simple parts of a sequence the TLD tracking can go uninterrupted for as long as 30-40 frames. For difficult parts with rapid movements or occlusions, the tracking can be restarted as often as needed. This showed to be a very fast and accurate way of ground truth annotation.

3.4 Tracking evaluation

Choosing a proper visual tracking evaluation metric is a very important step in the process of developing a new tracker. The visual tracking metric should be selected specifically for some particular tracking task (multi-target tracking for surveillance, 2D short-term single target tracking, 3D long-term single target tracking, etc.). An unbiased and easy to interpret metric provides valuable insights into a tracker performance. Ideally such metric should be used by everybody in the research community. However, currently, there is no standard widely accepted tracking metric either in the field of 2D tracking or in the field of 3D tracking. In this section I review the most popular metric on 2D and 3D single target tracking and justify the choice of the ones that I use in my research. All the experiments described in this thesis were performed on an Intel Core i5-4570, 3.20 GHz personal computer. The implementation language is C++.

All monocular and stereo datasets used in my experiments have a ground truth annotation. Therefore, I discuss only evaluation methods based on comparing the tracking results to the ground truth data. For details on how the ground truth data was obtained, see Section 3.3.

3.4.1 Evaluation metric for 2D tracking

The choice of a tracking evaluation metric depends greatly on the primary application of the tracker. The gole of this research is long term 3D whole body view face tracking. As the first step towards this goal, I developed a 2D face tracking algorithm that is able to track a person reliably and accurately in real time, distinguish this person from other people present in the field of view, reinitialize the tracking automatically if the person leaves the room and then comes back after some time. In other words, I chose a metric for 2D single target long-term tracking.

Often authors of generic single target tracking algorithms use some simple metric to compare their own tracker with other trackers. This is usually an average center location error (Xu Jia *et al.* 2012, Dinh *et al.* 2011, Ross *et al.* 2008) or average overlap

(Wei Zhong *et al.* 2012, Hare *et al.* 2011). The average overlap and average center position error are not very descriptive. When using these measures, it is hard to tell whether a tracker was very accurate to some point in a sequence and then failed and could not recover or it followed the target to the very end of the sequence, but not very accurately. In both situations, the resulting average center position errors can be quite similar. Other examples of simple measures used to compare trackers are tracking length (Kwon and Lee, 2009), failure rate (Kristan *et al.*, 2010), F-score (Smeulders *et al.*, 2014). They have the same disadvantage as the average center location error - they offer little insight into the tracker performance which limits their interpretability. Often quantitative comparison involves plotting the center location error or overlap versus frame number (Xu Jia *et al.* 2012, Ross *et al.* 2008). It is possible to use this plot to illustrate performance of several trackers on one particular sequence. Such plot can show at which frame each tracker failed. However, if the testing dataset consists of dozens of sequences, these plots are not very descriptive.

To overcome these problems with single value measures, several authors propose to visually compare tracking performance via performance summarization plots (Babenko *et al.* 2011, Wu *et al.* 2013, Salti *et al.* 2012). The two most notable examples are the precision plot and success plot, described in the next two paragraphs. Following Wu *et al.* (2013), I used these two plots for the 2D tracking evaluation in this research.

The precision plot (called "location error plot" in this thesis) shows the percentage of frames for which the estimated object location is within some threshold distance of the ground truth. The score for the threshold of 20 pixels is used as the representative score for each tracker. The threshold of 20 pixels was adopted from Wu *et al.* (2013). However, this threshold also has a special meaning for my dataset. For sequences with the 1280×1024 resolution, 20 pixels is approximately the average width of the tracked face. It is important to mention that my evaluation dataset contains videos with different resolutions. A location error equal to 20 pixels has a different impact in the 320×256 sequence compared to the 1280×1024 sequence. Therefore, when computing the location error plot, all errors are scaled in respect to the highest resolution used in the experiment (which is usually 1280×1024). For example, the 5 pixels error in the 320×256 sequence is multiplied by 4. This allows all different sequences to have equal impact to the final location error plot.

The success plot (called "overlap plot" in this thesis) uses the region overlap instead. The overlap score is defined as

$$S = \frac{|B_t \cap Bgr|}{|B_t \cup Bgr|}, \tag{3.2}$$

where B_t is the tracked bounding box, B_{gr} is the ground truth bounding box, $\cap$ and $\cup$ denote the intersection and union of two regions, respectively, and $|\cdot|$ denotes the number of pixels in the region. To measure the accuracy of a tracking algorithm on a sequence of frames, I count the number of frames whose overlap S is larger than the given threshold. The overlap plot shows the ratios of successful frames at the thresholds varied from 0 to 1. As the representative score for each tracker, I use the area under curve (AUC) of each overlap plot. Recently Čehovin *et al.* (2015) proved that the area under curve is equal to the average overlap. Also Čehovin *et al.* (2015) performed experimental evaluation of correlation of different tracking performance metrics. Their experiment showed that the correlation between the average center location error and average overlap is high. This is also true for the location error plot and the overlap plot. In my research in most cases these two plots look very similar with rare exceptions. For an example when the location plot and overlap plot look different, see Section 4.4.7).

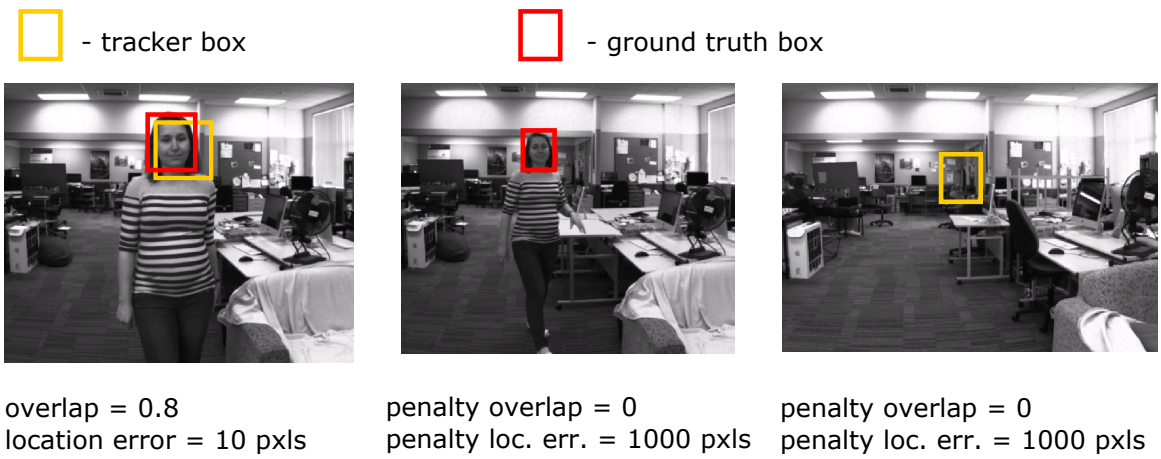

Figure 3.6: If the tracker outputs the target bounding box, when the target is not really visible, or the tracker cannot find the target, when the target is present in the image, then the penalty overlap and penalty location error are added to the final result.

In Wu *et al.* (2013) the location error plot and overlap plot are computed only for the frames where the ground truth bounding box is present and the evaluated tracker outputs a bounding box. This way the frames where the tracked target leaves the

camera field of view are not taken into account. To properly account for such frames, I use penalties described further. The following three situations depicted in Figure 3.6 are possible when evaluating tracking performance:

- A tracker outputs a bounding box and the ground truth box is available for this frame. Then the location error and overlap are estimated. If the boxes do not intersect, then the overlap is equal to 0.
- A tracker outputs a bounding box, but the target is not in the image and there is no ground truth box for this image. It means that the tracker is wrong and the penalty location error equal to 1000 pixels and penalty overlap of 0 are added.
- A tracker does not output a bounding box, but the target is present in the image and there is a ground truth box in this image. It means that the tracker is wrong and the penalty location error equal to 1000 pixels and penalty overlap of 0 are added.

For the location error, the penalty equal to 1000 pixels is chosen because it is a round number and it is close to the highest resolution in my dataset which is 1280×1024 pixels. This way the penalty error is similar to the highest location error that can occur in my testing dataset.

Another important issue that is often addressed in different metrics is how long a tracker can track the target. If in an experiment a tracker is initialized once in the beginning, it can fail in the beginning and the remaining part of the tracking results can be considered irrelevant. To solve this problems some authors propose to restart a tracker when it fails and count the restarts (Kristan *et al.* 2015, Kristan *et al.* 2016, Wu *et al.* 2015). Another approach is to evaluate each tracking algorithm numerous times from different starting frames across an image sequence (Wu *et al.* 2015, Wu *et al.* 2013). However, for my goal of long-term tracking, it is important for the tracker to be able to restart itself. In long term tracking, the tracked person can leave the cameras' field of view and come back after some time. The tracker should be able to cope with such a situation. Therefore I decided to initialize the tracker only once in the start of each sequence and see how it performs.

3.4.2 Evaluation metric for 3D tracking

My final goal is to do real time long-term 3D face tracking in the stereo feed coming from two cameras with an overlapping field of view. The system is set up so that the whole

body of the tracked person is visible most of the time. There is not much research in this area and there is no widely used metric for single target long-term 3D tracking using cameras with an overlapping field of view. Zoidi *et al.* (2014) measures the Stereo Frame Detection Accuracy (SFDA), which is the average overlap area between the tracked object in the left and right frame and the corresponding ground truth. Kobayashi *et al.* (2006) and Rougier and Meunier (2010b) propose to measure the average Euclidean distance in cm between the estimated position and the 3D ground truth. Many papers describing new 3D tracking algorithms perform qualitative estimation of 3D tracking results (for example Zoidi *et al.* (2013)). The 3D tracking field is so diverse and undeveloped, that I could not find any papers where the authors compare their own 3D tracking algorithm to 3D trackers published by other researchers.

As there is no standard metric for 3D tracking, I decided to adapt the metric that I use for 2D tracking (the location error plot and overlap plot) described in the previous section. However, in the case of 3D head tracking the overlap is a strong cue itself and can be used to improve tracking accuracy. Section 5.2.1 describes how to use the overlap to check that the 3D result is consistent. Therefore, I use the location error plot to measure the 3D tracking performance.

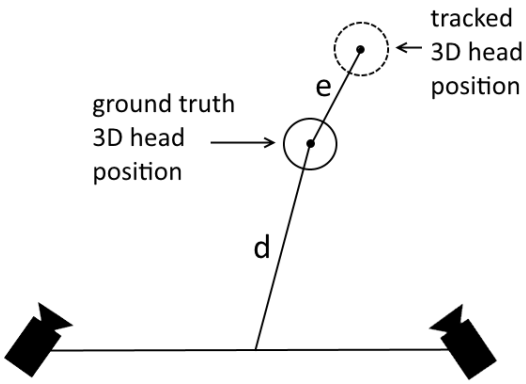

Figure 3.7: The relative 3D error is a ratio of the absolute location error e and the distance between the true target position and the midpoint of the cameras d.

I tried three different ways of computing the location error plot for 3D tracking:

- **Absolute 3D location error** in cm. To compute this error, the 3D ground truth position of the tracked head is computed by triangulating the center of the ground truth face bounding boxes. Then the 3D euclidean distance between the

triangulated 3D ground truth position and the 3D position output by the 3D tracker is estimated. As the representative precision score for each tracker, I use the score for the threshold of 20 cm. This threshold is chosen because it is similar to an average head size.

- **Relative 3D location error**. It is harder to track the head when it is further away, so the errors in the near distance tracking are more important than the errors in the long distance tracking. To take this into account, I compute the relative 3D location error as $L = \frac{e}{d}$, where e is the 3D location error and d is the distance between the ground truth 3D head position and the middle point between the cameras (see Figure 3.7). The relative 3D location error plot shows the ratios of successful frames at the thresholds varied from 0 to 1. The area under curve (AUC) of each plot is used to rank the tracking algorithms.

- **2D location error** in pixels averaged for the two cameras views. The absolute 2D error is computed as $L = 0.5(L_1 + L_2)$, where L_1 is the left camera 2D error in pixels, and L_2 is the right camera 2D error in pixels. As the representative score for each tracker, I use the score for the threshold of 20 pixels (similarly to the 2D tracking metric).

Examples of all three plots are shown in Figure 3.8. The three plots are very similar and there is no particular difference in using any of them. I decided to use the absolute 3D location error because the error is measured in cm and it is easier to interpret this error for real life 3D tracking applications than the averaged 2D error in pixels or the relative 3D error.

The absolute 3D location error is estimated only when the ground truth 3D location can be computed (the tracked target is visible in both views) and the evaluated 3D tracker was able to compute the 3D position. The penalty error equal to 1000 cm is added in two cases. The first case is when the ground truth 3D position cannot be computed, but the evaluated tracker returns the 3D position. The second case is when the ground truth 3D position is available, but the tracker does not return the 3D position because it failed in either of the views or in both views. The penalty error of 1000 cm is chosen because it is large enough to represent complete 3D tracking failure.

For 3D tracking similar to 2D tracking, I do not estimate robustness or restart the trackers when they fail. My experimental videos include a lot of full occlusions and other difficult scenarios. The trackers need to be able to recover by themselves.

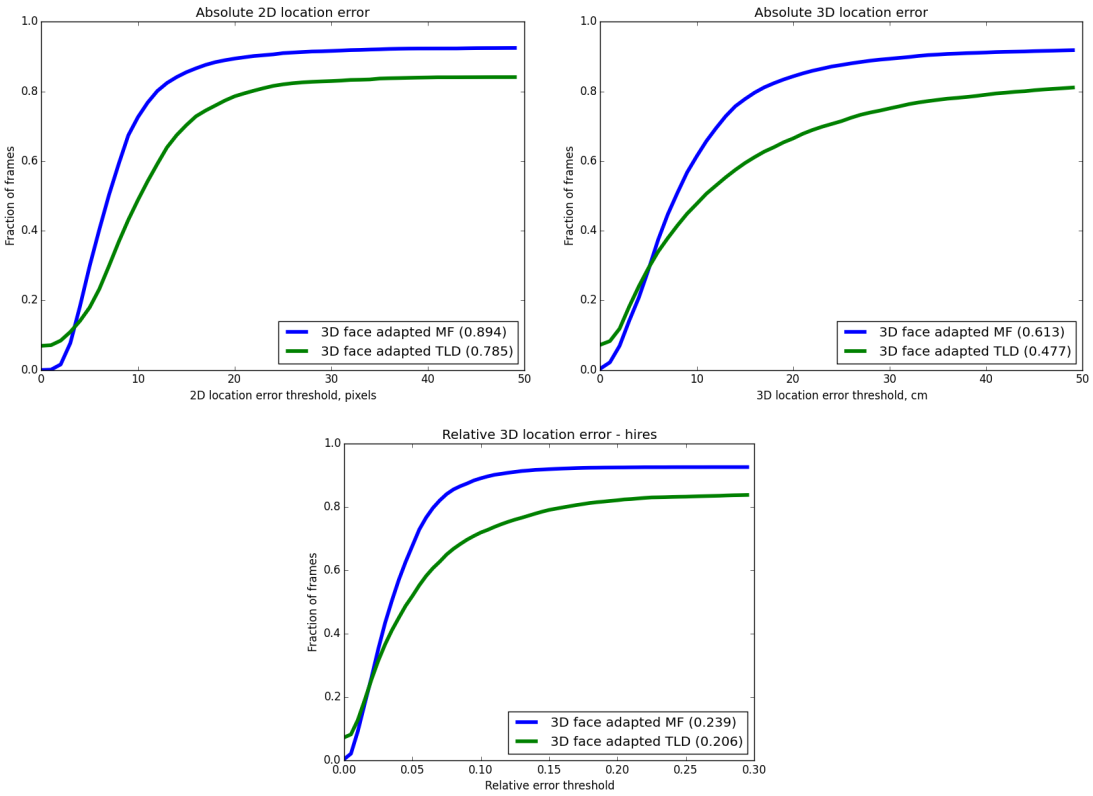

Figure 3.8: Three different location error plots (the 2D location error plot and the 3D location error plot in the top row, the relative error plot in the bottom row) show the results of the experiment that compares 3D face adapted TLD and 3D face adapted median flow. This experiment is described in Section 5.1.

For the goal of 3D tracking for real time interactive applications such as interactive displays, a few more specific metrics could be introduced, such as the smoothness of tracking and how fast the tracker recovers from full occlusion when the target reappears in the field of view. This is not in the scope of this research, but can be included in future research.

3.4.3 Frame rate estimation

The tracking frame rate is very important for real time applications. According to the requirements set in Section 1.2, the frame rate of my final 3D face tracking algorithm should be as close to real time (30 fps) as possible. At the first glance, the frame rate estimation is straight forward. The frame rate is usually computed as the number of frames processed by a tracker in one second. This works fine when the tracker speed

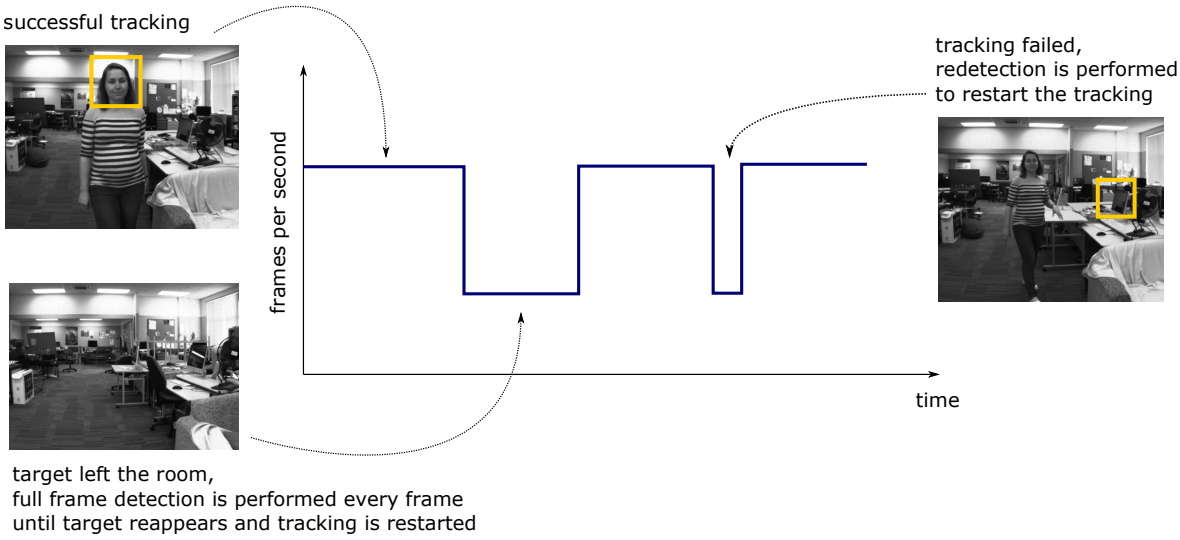

Figure 3.9: In the face adapted median flow tracker, the frame rate is slower for the frames when the target is lost and the whole frame detection is performed to restart the tracker.

is approximately constant through the sequence. However this is not the case for face adapted median flow described in this research. Face adapted median flow consists of two main parts: the median flow tracker (Kalal *et al.*, 2010b) and the Viola Jones face detector (Viola and Jones, 2001). Median flow is very fast and can run 30-60 fps depending of the image resolution. In contrast, the Viola Jones detector is pretty slow, running 1-10 fps depending of the image size. As explained in Section 4.4, in face adapted median flow the face detection on the whole image is performed only when the tracker lost the target. The rest of the time, the face detection is performed in two subwindows: in one random subwindow in the background and in one subwindow around the previous head position. Obviously, when tracking is successful, the area of search for the Viola Jones detector is smaller than the whole frame detection when the tracker has failed. Therefore, the tracking mode is considerably faster than the detection mode. The average frame rate of processing a sequence depends on how often and for how many frames the detection was performed (see Figure 3.9).

I tried two different ways of computing the frame rate for face adapted trackers: the average frame rate and the successful tracking frame rate. The average frame rate is the averaged value for the whole sequence. The successful tracking frame rate is the speed of tracking when tracking is successful. For most of my experiments I recorded both the average frame rate and the successful tracking frame rate. However, the average

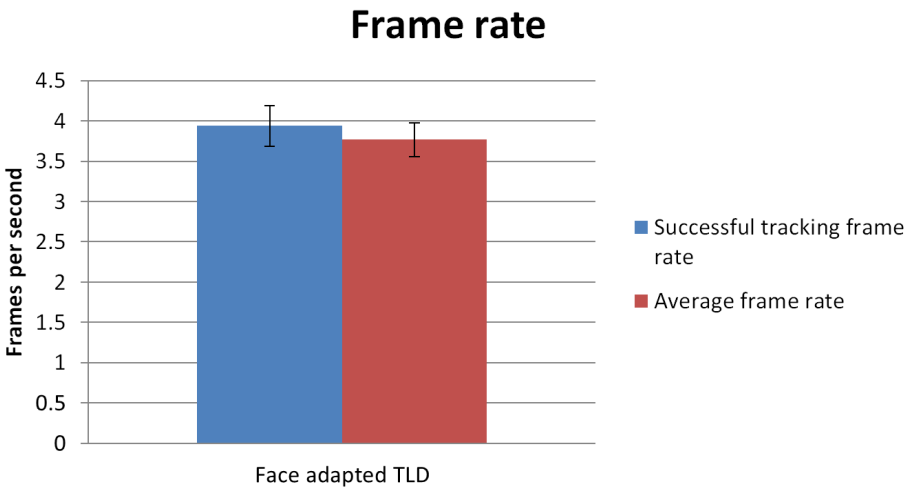

Figure 3.10: The successful tracking frame rate and the average frame rate for face adapted TLD for 37 high resolution videos. The two frame rates are very similar in this experiment.

frame rate was very close to the successful tracking frame rate most of the time. For example for the experiment described in Section 4.4.7, I evaluated face adapted TLD on 37 high resolution sequences and computed the successful tracking frame and the average frame rate. The two frame rates are shown in Figure 3.10. Even though they might differ substantially for some of the videos, they are very similar for the whole dataset. These two frame rates are very similar for other experiments as well. As there is not much difference between the two frame rates, I report the average frame rate in the following chapters.

3.5 Conclusion

In this chapter I described the tracking evaluation framework that I developed. It includes the video datasets with ground truth annotation and the metrics for 2D and 3D tracking evaluation. As there are very few publicly available datasets that are suitable for my research task, I recorded two large datasets: one using high resolution cameras with standard lenses and one using low resolution cameras with fisheye lenses. For creating datasets I carefully chose the content of the sequences and the challenging factors that are present in the videos. The developed tracker needs to be able to overcome some difficulties, but it is not expected to work in an arbitrarily difficult environment. I described how the TLD tracker can be used for semiautomatic ground truth annota-

tion. I chose fair and easy to interpret metrics for 2D and 3D tracking. The location error plot and overlap plots are used to visualize different trackers' performance. This evaluation framework can now be used for developing 2D and 3D trackers. The 2D tracker is discussed in Chapter 4. In Chapter 5 I talk about the 3D tracker.

Chapter 4

2D Face Tracking

The goal of this research is developing a real-time long-term 3D face tracking algorithm for whole body view videos. As the first step towards this goal, in this chapter I develop a 2D long-term face tracking algorithm.

This chapter is structured as follows. In Section 4.1 I explore the existing generic trackers to find out which one works best on the face tracking datasets that I collected. TLD and the context tracker show the best performance. Section 4.2 studies ways of speeding up TLD for working on high resolution video. In Section 4.3.3 I develop a novel approach of adapting generic trackers for tracking faces. Also this section compares different face adapted trackers. For the goal of face tracking for interactive displays, face adapted median flow shows the best results. In Section 4.4 several experiments are performed to find the best parameters for face adapted median flow.

4.1 Choosing the best public tracker

In this section, one of the contributions of this thesis is reported. The described experiments aim to find out which one of the publicly available generic tracking algorithms is best for long-term whole body view face tracking. It is hard to fairly compare different trackers and determine which one is the best due to the difficulty of choosing unbiased testing sequences. Often due to implementation specifics, a tracker can perform very well on one set of videos and fail on a different testing set. For example, trackers that perform very well on short videos, sometimes fail and cannot recover on long sequences. Therefore, to find the best existing tracker for some particular task, a dataset of videos specifically for this task should be collected. A dataset was collected to specifically evaluate long-term whole body view face tracking (for more details see Section 3.1).

The testing framework described in Wu *et al.* (2013) was used to compare the existing generic trackers on our dataset. Wu *et al.* (2013) provide the executables for 29 publicly available generic trackers with uniform input and output formats to facilitate large scale performance evaluation. In this experiment, I managed to evaluate only 17 out of 29 trackers. The other 12 trackers do not work properly on some of our videos. It is not possible to fix them because their source code is not available. However, it was possible to test the 5 trackers (SCM (Wei Zhong *et al.*, 2012), Struck (Hare *et al.*, 2011), ASLA (Xu Jia *et al.*, 2012), TLD (Kalal *et al.*, 2011), CXT (Dinh *et al.*, 2011)) that showed the best results in Wu *et al.* (2013), and many other trackers as well.

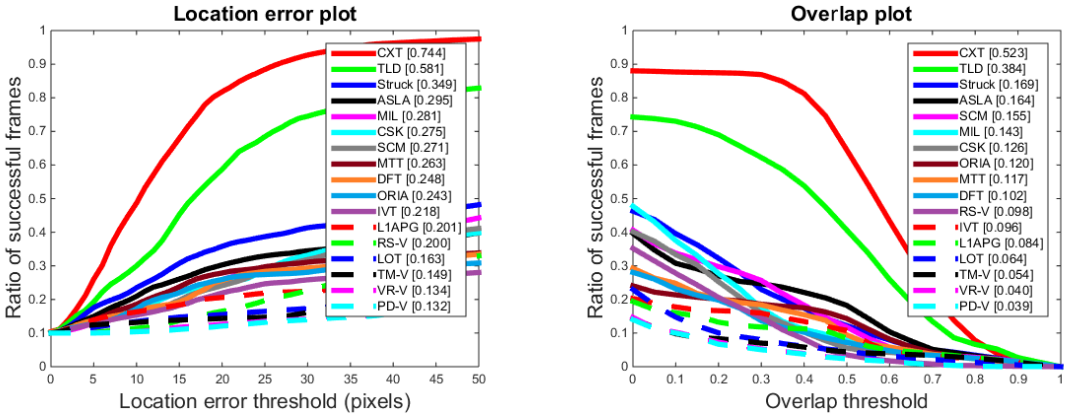

Figure 4.1: The performance of the generic tracking algorithms on our dataset.

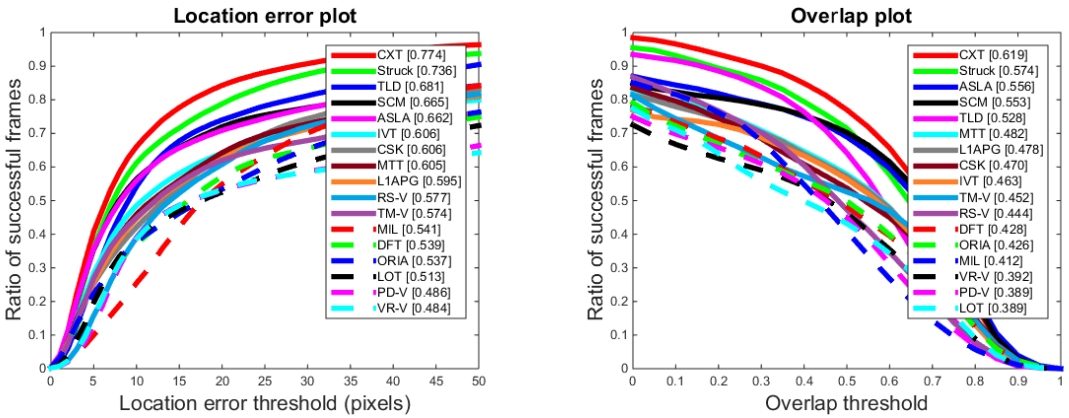

Figure 4.2: The performance of the generic tracking algorithms on the public face tracking dataset.

Figure 4.1 shows the performance of the generic trackers tested on my high resolution gray scale dataset. There are two clear leaders that perform much better than the

other trackers: TLD and the context tracker (CXT). Figure 4.2 shows the performance of the same trackers on the public face tracking dataset from Wu *et al.* (2013). The context tracker, Struck and TLD are leading on this dataset. Unlike the performance on my dataset, in the case of the public dataset, there is no large gap between the leading trackers and the rest of the trackers. Also it is interesting to note that two other trackers (ASLA and SCM) that show strong performance in the public dataset from Wu *et al.* (2013), perform considerably worse on our dataset. This can be explained by the fact that trackers that perform very well on short sequences, can fail on long videos. All public videos are around 500 frames long, while videos in our dataset are 1500-3000 frames long. When running on long videos, the ability of a tracker to detect a failure and reinitialize itself plays an important role. The context tracker and TLD have very strong failure detection and reinitialization procedures. That is why they have a clear advantage on our long-term sequences.

The context tracker and TLD show the best performance both on our dataset and on the public dataset. The context tracker is based on TLD and uses similar principles. Unfortunately, the authors of the context tracker do not provide the source code. For this reason, TLD was chosen for developing the long-term whole body view face tracker. The C++ implementation of TLD by George Nebehay (Nebehay, 2012) was used. The Nebehay implementation of TLD has the foreground segmentation as an additional stage in the cascade detector. Nebahay claims that his implementation is 3 times faster than the original MatLab implementation and can be even faster for a GPU implementation.

The experiments described in this section also show that the choice of video sequences for testing tracking algorithms can drastically affect the experimental results.

4.2 Speeding up TLD

In the experiment described in the previous section TLD shows outstanding performance on our datasets. This section investigates two ways of speeding up TLD. The average frame rate of running TLD on our high resolution video sequences (1280×1024) is around 5 fps. For stereo tracking the speed is even slower. It's important to speed up the tracker to get the performance as close to real time (30 frames per second) as possible.

4.2.1 Background subtraction

TLD performs tracking and detection on every frame. On each frame around 40 ms is spent on tracking and around 150 ms is spent on detection. One way to speed up TLD is to limit the search area of the TLD detector. If the background is stable, this can be done by background subtraction.

Figure 4.3: Background subtraction. The white rectangles show the segmented foreground. The yellow rectangle shows the tracked target.

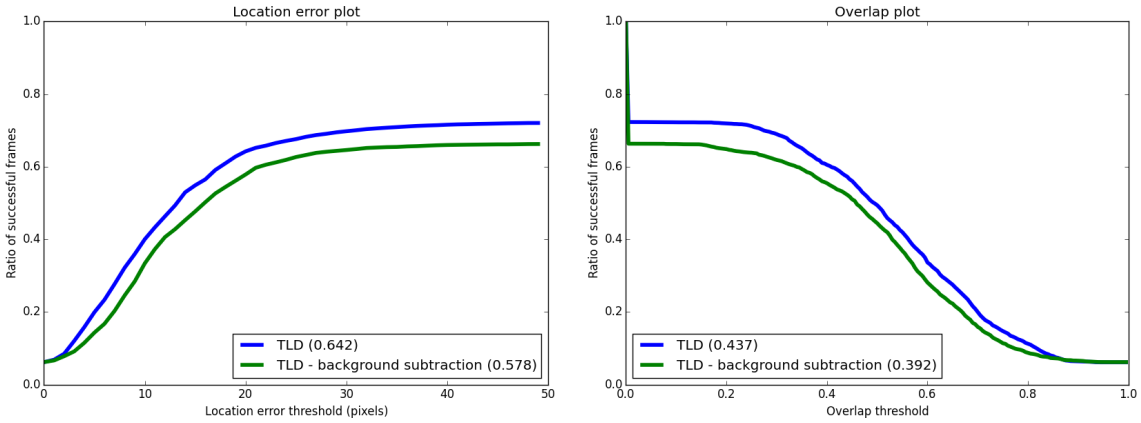

Figure 4.4: The performance of TLD with background subtraction and without it.

There are many different methods of background subtraction. The survey of these methods can be found in Radke *et al.* (2005). I used the method implemented by

Nebehay (2012). In this implementation, background subtraction is performed in four steps:

1. The absolute difference between the background image I_{bg} and the current image I is calculated:

$$I_{\text{absDiff}} = |I_{\text{bg}} - I| \tag{4.1}$$

2. Then a threshold of 16 is applied to get a binary image:

$$I_{\text{binary}}(x, y) = \begin{cases} 1 & \text{if } I_{\text{absDiff}}(x, y) > 16 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

3. Then the labelling algorithm described in Chang and Chen (2003) is applied. This algorithms computes contours of all connected components and their bounding boxes.

4. The components whose bounding boxes are smaller than the original selected box are discarded.

After background subtraction is finished, the search for the target is performed in the segmented area only. Figure 4.3 shows a successful background segmentation.

To compare TLD performance with background subtraction and without it, I conducted an experiment on 10 prerecorded sequences with a background image available. The results are shown in Figure 4.4. The accuracy of the TLD tracker with background subtraction is worse because foreground segmentation is not always correct. Also when the detector search area is limited, some of the false positive examples are not detected and the detector learning is not as effective. The average frame rate of TLD with background subtraction is **6.4 fps**. The average frame rate of TLD without background subtraction is **4.8 fps**. The speed-up is not significant considering the accuracy loss. Therefore, it was decided to try other methods for speeding up TLD.

4.2.2 Subwindow detection

Another way to speed up TLD by limiting the detection area is investigated further. TLD performs tracking and detection on every frame. If tracking is not successful, detection is used to reinitialize the tracking. If tracking is successful and confident, the

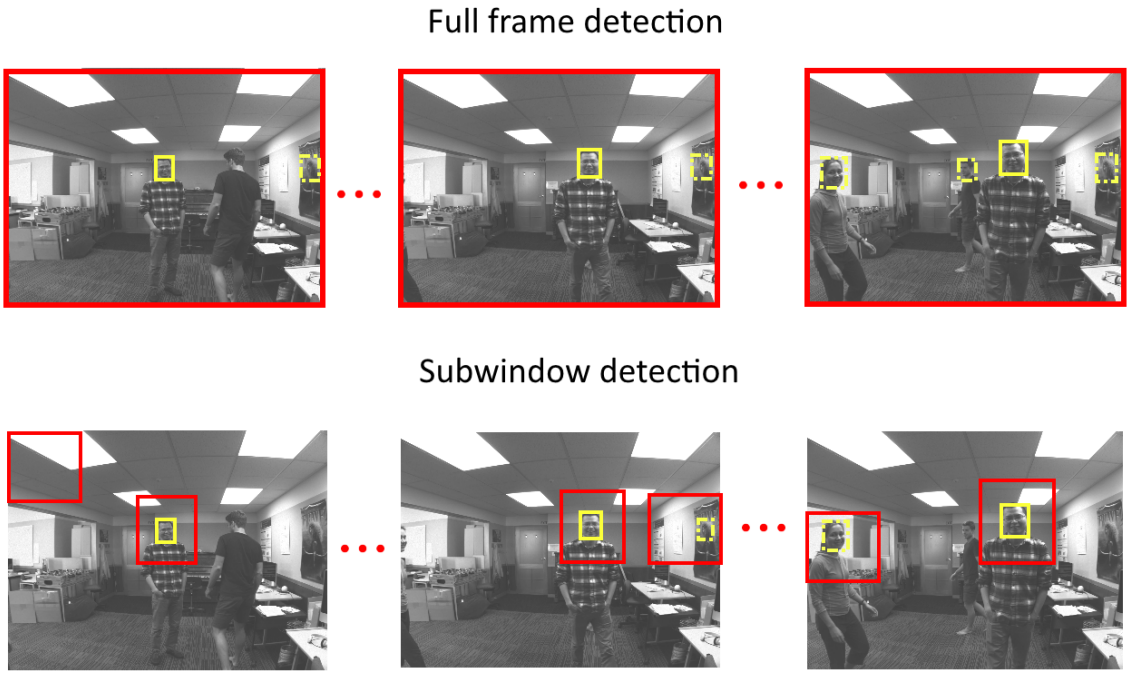

Figure 4.5: The subwindow detection. Red boxes show the detection area. Yellow solid line boxes show the tracked target. The dashed yellow boxes show the false positive examples.

results of the tracker and the detector are used for learning. The structural constraints of the data together with the tracker and detector responses are used to find false positive detections. These false positive results are used as negative examples for updating the object model in the detector. Most of these negative examples are patches in the background that have similar appearance to the tracked target. Therefore, if the background is stable, these patches usually appear in the same places for several frames in a row and are learned by the detector several times. Consequently, there is no need to perform the whole frame detection every frame. When the tracker and the detector are confident in one frame, in the next frame it is possible to perform detection around the previous target position. This will help to correct the tracker if it starts to drift away from the target. Also, it is important to scan some part of the background to check if any new false positive examples have appeared. The detection subwindow in the background changes its position every frame. In a number of frames the whole image area is scanned. If the detector and the tracker are not confident, then the whole frame detection is performed in the next frame.

Figure 4.5 illustrates the subwindow detection method. The top row shows the full frame detection. All the false positives are found every frame (the dashed yellow boxes

around other people's faces and a face-like patch in the background). The bottom row shows the subwindow detection. The true positive (the tracked target) is found in every frame because the detection is performed in a subwindow around the target location in the previous frame. The false positive examples are found when the subwindow detection is performed in that region. The subwindow detection is performed on every part of the image at least once in 10 frames.

The experiment described further investigates how the subwindow detection method affects the speed and accuracy of TLD. In TLD all the subwindows are defined during the initialization step in the beginning. These subwindows are divided into 10 equal size batches and the search is performed on one of these batches at a time. The TLD detector with the subwindow detection is called FastTLD. Figure 4.6 shows the results of the experiments. Both methods were checked on high resolution sequences (1280×1024 pixels) and on low resolution sequences (640×512 pixels) separately. The corresponding frame rates are provided in Figure 4.7.

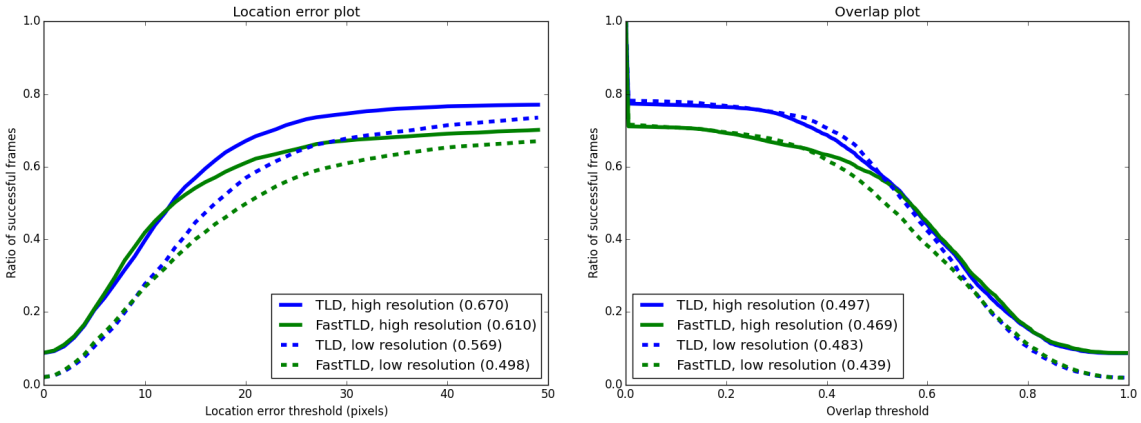

Figure 4.6: The performance of TLD and FastTLD on high resolution sequences and on low resolution sequences.

This experiment showed that the FastTLD accuracy is worse than the TLD accuracy. Also both methods perform better in high resolution images compared to the performance on low resolution images. Using FastTLD and low resolution sequences helps to speed up the performance significantly (24 fps versus 8 fps) and gets close to real time.

4.3 Adapting for face tracking

Note: Some portions of this section were published in Mikhisor *et al.* (2015).

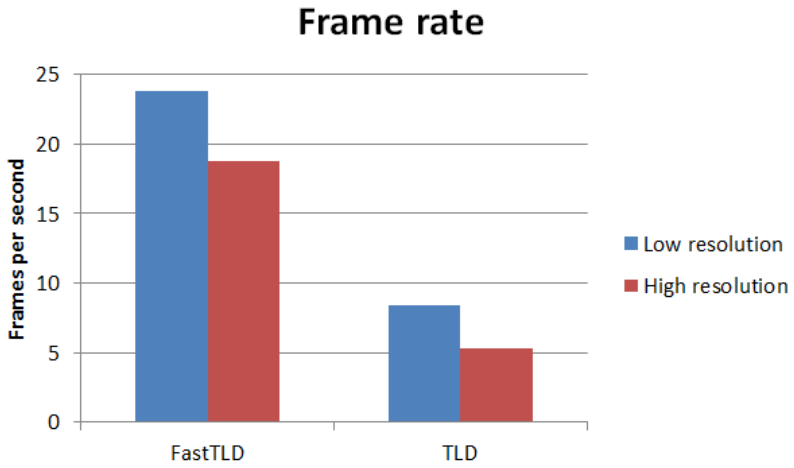

Figure 4.7: The frame rates of TLD and FastTLD in low resolution images and in high resolution images.

If we employ a generic tracker to track a face, we do not use one very important source of information: we know what a typical face looks like. This information can improve tracking considerably. In this section, I first discuss the face adaptation of TLD called FaceTLD (Kalal *et al.*, 2010a) developed by the author of TLD. Then I describe my adaptation of a generic tracker for tracking faces using the Viola Jones face detector (Viola and Jones, 2001) and the structural constraints of the video volume. This face tracking adaptation can be used with any generic tracker. Also it is possible to adapt a tracker to track any other category of subjects such as cars or human figures if the corresponding offline trained detector is available. I performed experiments with the three generic trackers: MIL, TLD and Struck. For all the three trackers, the face adaptation shows substantial tracking accuracy and robustness improvement.

4.3.1 FaceTLD

It is possible to increase tracking accuracy by giving up generality. TLD shows great performance because it has very strong procedures to learn the changing appearance of the tracked subject and to reinitialize tracking when it fails. However, even TLD fails in longer sequences with difficult scenarios such as rapid movements, a confusing background, or when the tracked subject goes in and out of the camera's view. Due to occlusion or rapid movement, the median flow tracker used in TLD can drift away from the target and the TLD detector starts to learn false examples and its reliability deteriorates. By restricting the domain of possible targets to a certain type of object

(faces in our case), it is possible to improve TLD or any other generic tracker greatly.

Figure 4.8: The Viola Jones face detector finds many false positive examples.

The face adaptation method described in this research was inspired by the face adaptation of TLD, called FaceTLD (Kalal *et al.*, 2010a). In FaceTLD, the online TLD detector is replaced by an offline trained face detector and a validator. The authors used their own offline trained face detector (Kalal *et al.*, 2008). The offline trained detector localizes frontal faces and the online trained validator decides which faces correspond to the tracked subject. The validator consists of a collection of positive and negative examples. To decide whether a new face found by the detector is the true target or not (another person's face or a face-like background patch), the validator compares this new detection with its collections of positive and negative examples.

In FaceTLD, the tracking is initiated in the first frame. This initial example is tracked by a tracker and all patterns along the trajectory are accepted as positive examples. Patches surrounding a validated trajectory are considered as negative examples.

According to Kalal *et al.* (2010a), FaceTLD is considerably more accurate than TLD. This method seems very promising. However, offline trained face detectors, such

as the Viola Jones detector, often have high recall and low precision. This means that usually they find most of the faces in an image and many face-like patches in the background. In the example shown in Figure 4.8, the Viola Jones face detector found the one true face and mistook five patches in the cluttered background for faces. By learning negative patches as the ones that surround the validated tracker's trajectory, these face-like background patches far away from the track are not taken into account. If the target is lost, and the detector tries to restart the tracking, it's hard to choose which one of the many detections is the true target.

The face adaptation method that I developed is inspired by FaceTLD but has several differences to FaceTLD. Firstly, it can be applied to any generic tracker. Secondly, it uses a different strategy of collecting positive and negative examples using the data structural constraints. This method is described in the next section. Unfortunately the authors of FaceTLD do not provide its source code and the dataset that they used for their experiments is not publicly available. Therefore, I cannot compare my face adaptation method described in the next sections to FaceTLD.

4.3.2 Using structural constraints for creating the face model

In this section, one of the main contributions of this thesis is described. I propose a novel method of adapting any generic tracker for tracking faces using a face detector and structural constraints in the unlabelled data. The face detector is executed in parallel with the tracker and then structural constraints are used to create the tracked face model in the space of the face detector results. This model is used together with the detector to correct or restart the tracker.

The data are structured if knowing the label of one example restricts the labelling of the others. For instance, in object detection, the task is to label all possible image patches of an input image either as positive (object) or as negative (background). A unique object can occupy at most one location in the input image. In a video, the object location defines a trajectory, which is illustrated in Figure 4.9. The trajectory represents a structure in the labelling of the video sequence. All patches close to the trajectory share the same positive label, patches far from the trajectory are negative. These structural constraints allow to label some of the unlabelled data. In TLD, the structured unlabelled data is used to update the TLD detector after processing each frame.

I propose to use structural constraints to create the tracked face model in the space of the face detector detections. In my work I use the Viola Jones face detector, although

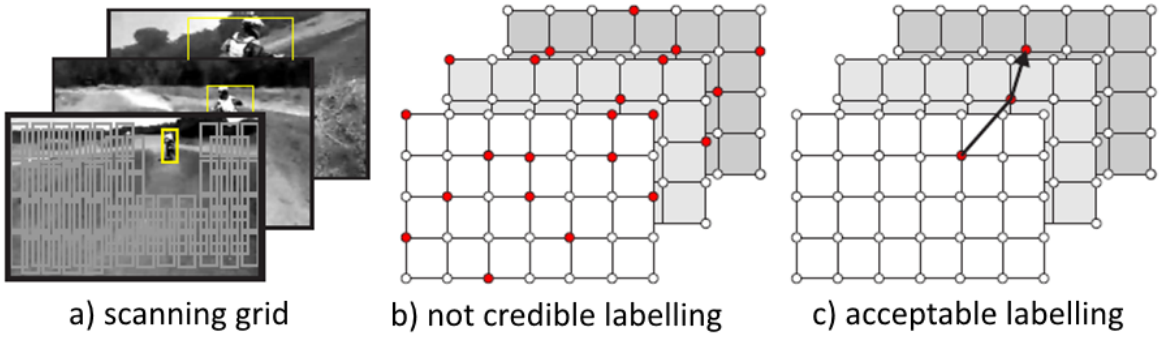

Figure 4.9: Illustration of a scanning grid applied to three consecutive frames (a) and corresponding spatio-temporal volume of labels with unacceptable (b) and acceptable (c) labelling. Red dots correspond to positive labels. The image is taken from Kalal (2011).

other detectors can be used (e.g. Liao *et al.* 2016, Mathias *et al.* 2014). This detector is learned offline using the supervised learning method. The detector is trained to find any face in an image. However, for the task of one person tracking, we need to ignore all other faces except the one that we are tracking. Another problem with the face detector is that it outputs many face-like background patches (see an example in Figure 4.8). It is not possible to retrain the Viola Jones face detector online the same way as the TLD detector is retrained every frame. For this reason, we create the tracked face model M in the space of all Viola Jones detections. The face model is a dynamic data structure that represents the object appearances and its surroundings observed so far. It is a collection of positive and negative patches. Positive patches are the true target face detected by the detector. Negative patches are other faces and face-like background patches detected by the detector. The tracked face trajectory in the video volume and the corresponding trajectory in the Viola Jones detections space is shown in Figure 4.10.

The same way as in TLD, the relative similarity measure S^r is used throughout the system to indicate how much an arbitrary patch P resembles the face appearance represented in the face model M. To compute the relative similarity S^r, we first compute the similarity with the positive nearest neighbour $S^+(P, M)$ and the similarity with the negative nearest neighbour $S^-(P, M)$:

$$S^+(P, M) = \max_{P_i^+ \in M} S(P, P^+), \tag{4.3}$$

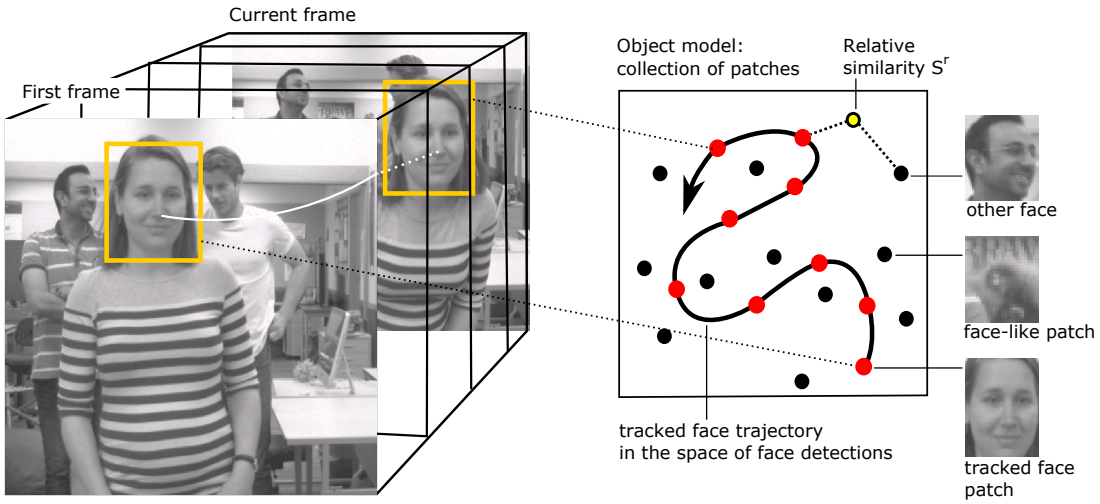

Figure 4.10: Illustration of a trajectory in video volume and corresponding trajectory in the appearance space.

.

$$S^-(P, M) = \max_{P_i^- \in M} S(P, P^-). \tag{4.4}$$

The similarity between two patches P_i, P_j is defined as

$$S(P_i, P_j) = \frac{1}{2}(\mathrm{NCC}(P_i, P_j) + 1), \tag{4.5}$$

where NCC is a Normalized Correlation Coefficient (Rodgers and Nicewander, 1988). The similarity ranges from 0 to 1. The relative similarity is computed as

$$S^r = \frac{S^+}{S^+ + S^-}. \tag{4.6}$$

Relative similarity ranges from 0 to 1, higher values mean high confidence that the patch depicts the object.

The face model M is built up when tracking and detection are successful. It is used to restart the tracking when it fails. The implementation details follow in the next subsection.

4.3.3 Face adaptation implementation

Any tracker can be adapted for tracking faces using the proposed method described here. A tracker is treated as a black box which receives an initialization rectangle in

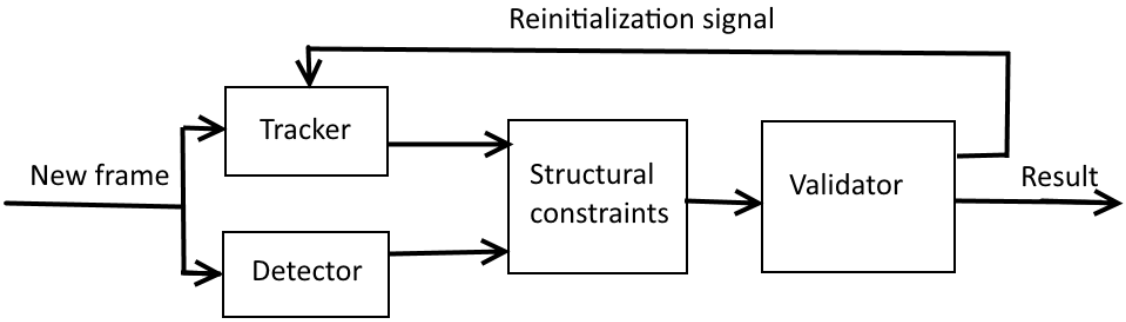

Figure 4.11: The block diagram of the face tracking adaptation method.

the first frame, and then for each subsequent frame it outputs the target's bounding box. The only requirement is that the tracker can be reinitialized when needed. The block diagram of the face tracking adaptation method is shown in Figure 4.11. This gives a wide variety of trackers that can be adapted for face tracking using this method. It allows a combination of all the best features of any state-of-the-art tracker and a reliable face redetection system.

The system consists of the main parts:

- **Tracker**. Any generic tracker that can track a target from frame to frame.
- **Detector**. An offline trained detector learned to detect some domain of objects. I use the Viola Jones detector to detect faces. However, it is possible to train and use any other detector, for example a car detector (Yebes *et al.*, 2014) or other sort of face detector (e.g. Liao *et al.* 2016, Mathias *et al.* 2014).
- **Structural constraints**. The temporal and spatial structure of the video volume is utilized as the structural constraints. The constraints are used to build the tracked face model in the space of the face detector results.
- **Validator**. This module validates the final tracking result for the current frame or sends a reinitialization signal to the tracker.

The processing of each frame consists of 4 main steps shown in Alg. 1. Each step is explained in detail further.

Algorithm 1 Face adapted tracking

Input: Current image I_t

Output: The target face bounding box *trackerBox*

1: *Tracker* $\rightarrow$ *trackerBox*

2: *Detector* $\rightarrow$ *faceCandidates*

3: *(trackerBox, faceCandidates)*$\rightarrow$ *StructConstraints* $\rightarrow$ *(posCandidates, negCandidates)*

4: *(posCandidates, negCandidates)* $\rightarrow$ *Validator* $\rightarrow$ *result*

Tracking

The first step is tracking that is performed on every frame by the chosen tracker. The tracker returns a rectangle which is the tracked face bounding box if the tracker is right, or a box somewhere in the background or around another person's face if the tracker has failed.

Face detection

At the second step, face detection is performed. If frames are small and the detector is fast, detection can be performed on every frame. Otherwise, it can be done periodically and when the tracker fails. The face detector often outputs several face candidates if there are other people in the camera viewing zone or face-like background patches.

Applying structural constraints

After the tracking and detection is finished, the structural constraints are applied to the tracking and detection results (Alg. 2). The temporal and spatial constraints are used for processing of labelled and unlabelled data. In this situation the tracker result can be considered as a labelled patch because the tracker is supposed to be tracking the target face. However, the tracker can be wrong. The detector outputs several patches which can be faces or face-like background. We do not know in advance which one of these patches is the tracked face. Therefore, we can say that the detector outputs unlabelled data. After applying the structural constraints, the tracker and detector results are labelled as positive examples (examples of the tracked face) and negative examples (face-like background or other faces).

Algorithm 2 Applying structural constraints

Input: *trackerBox, faceCandidates*

Input: *threshold_1* (0.3)

Output: *posCandidates, negCandidates*

Output: *posExamples, negExamples*

clear *posCandidates*
clear *negCandidates*
for each *faceCandidate* in *faceCandidates* **do**
    if *overlap (faceCandidate, trackerBox)*> *threshold_1* **then**
        Add *faceCandidate* to *posCandidates*
    else
        Add *faceCandidate* to *negCandidates*
    end if
end for

if $|posCandidates| == 1$ **then**
    Add *posCandidates* to *posExamples*
    Add *negCandidates* to *negExamples*
end if

The temporal structure of the video volume implies that the object moves on a smooth trajectory. The tracker estimates the trajectory. If the tracker box overlaps with one of the detection boxes, this is a positive example and it is added to the collection of positive examples. The overlap is defined as

$$S = \frac{|B_t \cap Bgr|}{|B_t \cup Bgr|}, \tag{4.7}$$

where B_t is the tracked bounding box, B_{gr} is the ground truth bounding box, $\cap$ and $\cup$ denote the intersection and union of two regions, respectively, and $|\cdot|$ denotes the number of pixels in the region.

The spatial structure of the video volume implies that the object can appear at a single location in a single frame only. When one of the detection results is marked as positive, all the other detections in the frame are marked as negative examples and added to the collection of negative examples.

If the tracker box does not overlap with any of the detection boxes, then the tracker and detection are considered unreliable. No positive or negative results are added to

the collections and the tracker needs to be restarted.

An example of applying structural constraints is shown in Figure 4.12. In this image, one of the detector results has a high overlap with the tracking result. This result is saved as a positive example. All the other detections are saved as negative examples.

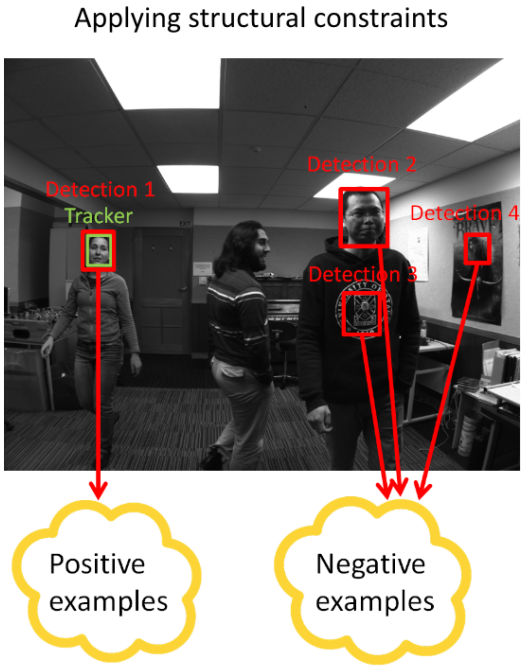

Figure 4.12: When tracking and detection is successful, the structural constraints are applied to the results.

In Alg. 2, *threshold_1* is used to determine weather an overlap of a face detection and a tracking result is large enough to add this face detection to positive candidates. I found the optimal value of this threshold experimentally. The results for different threshold values for face adapted TLD are shown in Table 4.3.3. The default value of this threshold was set to 0.3.

Result validation

After the structural constraints are applied, the result validation happens (see Alg. 3). The validator evaluates the positive and negative examples and either outputs the tracker bounding box as the final target face position for this frame or initiates the tracker reinitialization. There are two cases when the tracker reinitialization can happen.

threshold_1	**Average overlap**
0.05	0.628
0.1	0.627
0.2	0.629
0.3	**0.629**
0.4	0.629
0.5	0.583

Table 4.1: Average overlap of the face adapted TLD tracking results and the ground truth boxes for different values of *threshold_1*. In bold is the default value that was used in all other experiments.

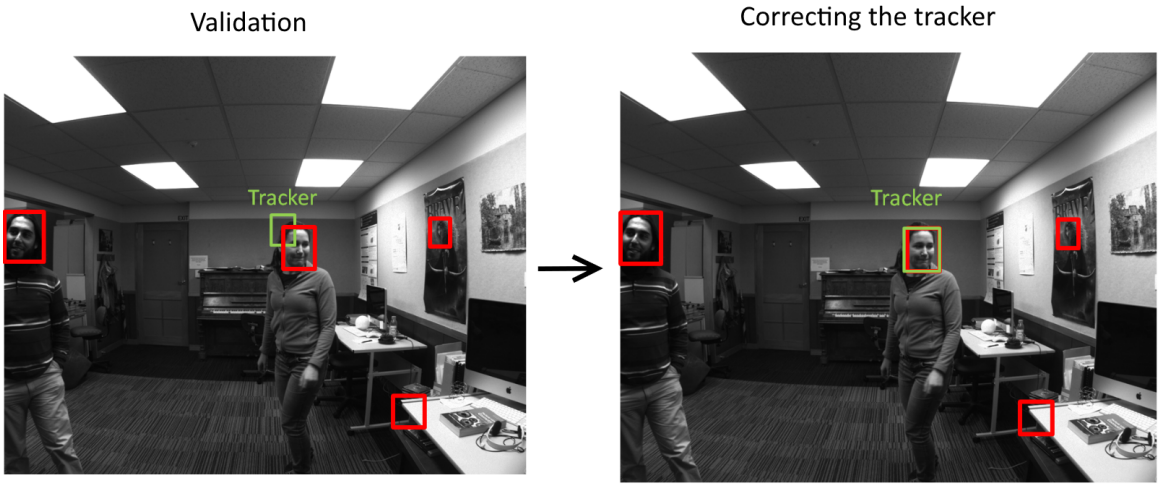

Figure 4.13: If the tracker and the detector results overlap is lower than a threshold, the tracker is reinitialized.

In the first case, the tracker box overlaps with one detection box and therefore there is exactly one positive candidate, but the overlap between the detection box and the tracker box is small. This means that the tracker is still tracking the target but started to drift away. In this case the reinitialization helps to prevent drifting. The validator sends the positive example (the overlapping detection result) to the tracker. The tracking starts from the new position in the next frame. An example of this case is shown in Figure 4.13.

The second case when the tracker reinitialization can happen is when there are no positive candidates or more than one positive candidate. In other words, there is no

Algorithm 3 Result validation

Input: *trackerBox, posCandidates, negCandidates*

Input: *threshold_2* (0.8), *threshold_3* (0.3), *threshold_4* (1), *threshold_5* (0.6)

Output: *trackerBox*

```
1: if |posCandidates| == 1 then
2:     if overlap (posCandidate, trackerBox)< threshold_2 then
3:         trackerBox ← posCandidate
4:         reinitTracker()
5:     end if
6: end if
7:
8: if |posCandidates| == 0 OR |posCandidates| > 1 then
9:     find (faceCandidates) → [bestCand, bestScore]
10:    if overlap (bestCand, prevDetect)> threshold_3 then
11:        redetectCount += 1
12:    else
13:        redetectCount = 0
14:    end if
15:    if redetectCount >= threshold_4 AND bestScore > threshold_5 then
16:        trackerBox ← bestCand
17:        reinitTracker()
18:    end if
19:    prevDetect ← bestCand
20: end if
```

overlap between the tracker box and any of the detection results or the tracker box overlap with more than one detection result. In this situation it is assumed that the tracker has lost the target and needs to be reinitialized. An example of this situation is shown in Figure 4.14. To do that, the validator computes the relative similarity S^r (Equation 4.6) for each detection result and finds the one with highest similarity to the tracked face model. If the candidate with the highest similarity is temporally and spatially consistent (holds for at least two frames in a row), the validator assumes that this is the true target and reinitializes the tracker. The validator helps to distinguish the true target from the background and from other people's faces as well. The function *reinitTracker()* in Alg. 3 calls the tracker specific reinitialization procedure.

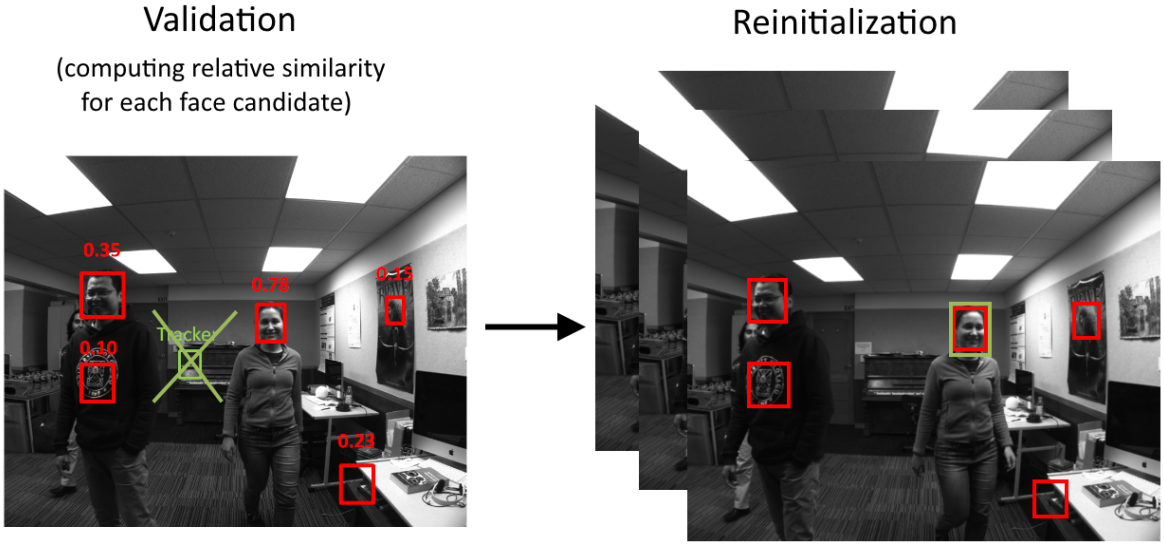

Figure 4.14: If the tracker and the detector results do not overlap, the tracker needs to be reinitialized. To reinitialize the tracker, the detection result with the highest similarity to the face model needs to hold for at least two frames.

Four different thresholds are used in Alg. 3. *threshold_2* is used to determine weather an overlap of the face detection and the tracking result is small enough to reinitilize the tracker. *threshold_3* is used to determine weather an overlap of the current face detection and the detection in the previous frame is large enough to believe that the same face was detected in this frame and in the previous frame. *threshold_4* is how many frames in a row the face should be redected to be able to reinitilize the tracker. *threshold_5* is the minimum relative similarity S^r that the best face detection needs to have to be able to reinitilize the tracker. All these thresholds were estimated experimentally. The results are shown in Table 4.3.3.

4.3.4 Adapting generic trackers for tracking faces

The described face adaptation method can be applied to any generic tracker if this tracker can be restarted on demand. I evaluated the method on three generic trackers: TLD[1] (Kalal *et al.*, 2011), Struck[2] (Hare *et al.*, 2011) and MIL[3] (Babenko *et al.*, 2011). According to the evaluation of 29 trackers by Wu *et al.* (2013) and a more

[1] https://youtu.be/8xYphOdOZbU
[2] https://youtu.be/LOWsRXVYUBI
[3] https://youtu.be/zZcZWW9elgM

threshold_2	**AO**	*threshold_3*	**AO**	*threshold_4*	**AO**	*threshold_5*	**AO**
0.4	0.564	0.05	0.629	0	0.604	0.3	0.628
0.5	0.610	0.1	0.629	**1**	**0.639**	0.4	0.629
0.6	0.630	0.2	0.629	2	0.637	0.5	0.622
0.7	0.578	**0.3**	**0.631**	3	0.633	**0.6**	**0.637**
0.8	**0.629**	0.4	0.631			0.7	0.582
0.9	0.593	0.5	0.626				
		0.6	0.620				

Table 4.2: Average overlap (**AO**) of the face adapted TLD tracking results and the ground truth data for different values of thresholds used in Alg. 3. The values used as default are shown in bold for each threshold.

recent extensive evaluation of 19 trackers by Smeulders *et al.* (2014), TLD and Struck are among the best 5 trackers and MIL is among the best 15 trackers. I chose these trackers because of their outstanding performance and because their implementations are available online. All three algorithms were used with the default parameters provided by the authors. For face detection I use the standard Viola Jones face detector provided in the OpenCV library. In this experiment the Viola Jones detector is used once in 25 frames or when the tracker loses the target.

The performance of TLD, Struck and MIL with and without the face tracking adaptation on public sequences is shown in Fig. 4.15 and on my sequences in Fig. 4.16. The face tracking adaptation improves tracking performance for all image sequences and all the three trackers. However, for my image sequences the face tracking adaptation of Struck and MIL shows a much greater improvement than TLD. This can be explained by the fact that Struck and MIL perform much better on short sequences, while on long ones they eventually drift away and are unable to recover. At the same time TLD has its own re-detection procedure, which helps to recover tracking on long sequences. That is why TLD shows similar performance on public short sequences as well as on my long sequences.

The results for testing the method on the public dataset are shown separately from the results of testing on our dataset to show that the dataset that I made is not biased. The face adaptation improvement is clearly visible on both my dataset and the public dataset. To prove that the results are significant, the standard deviation is shown by

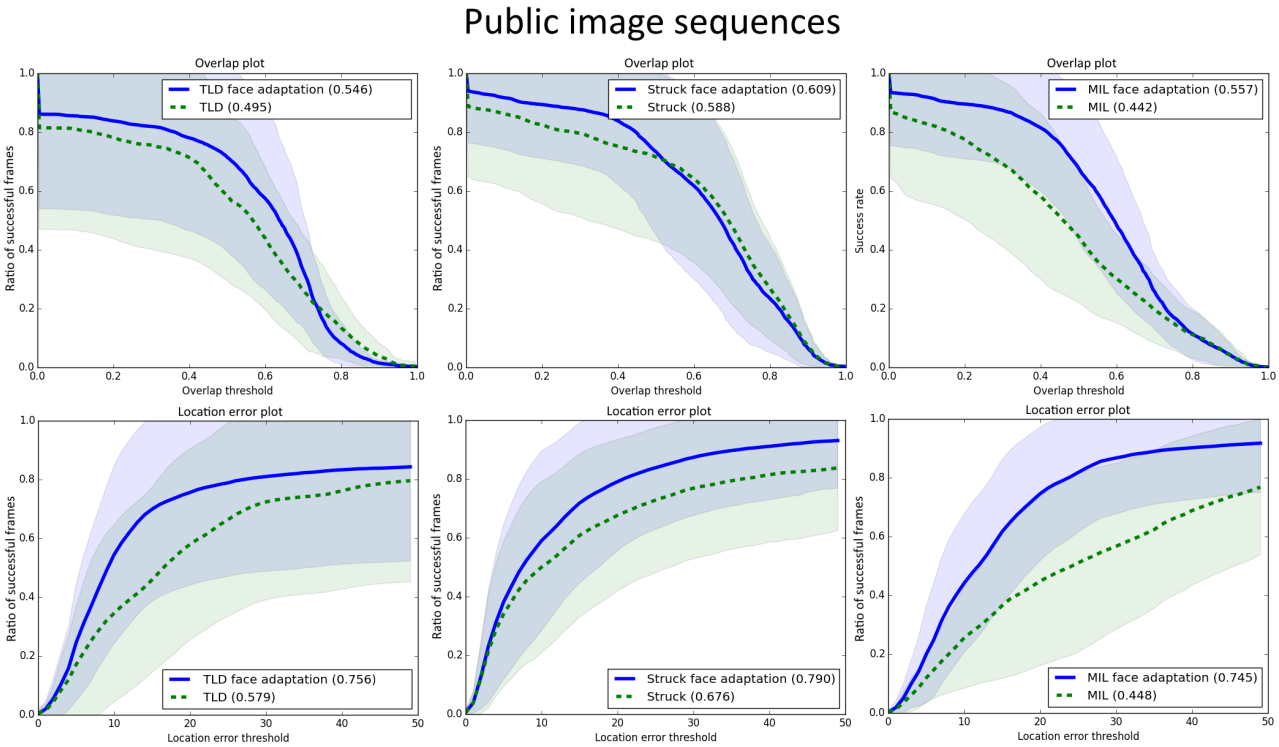

Figure 4.15: Performance evaluation on public image sequences. The overlap success plots for the three trackers are shown in the top row and the location precision plots are shown in the bottom row. Each line in a plot shows the mean performance over the image sequences and the shading shows one standard deviation.

shading in the plots in this subsection. In the following sections the standard deviation is not shown to keep the plots uncluttered.

Some examples of face tracking adaptation on public sequences are shown in Fig. 4.17. In this figure three difficult cases are shown. In the top row, the lecturer's face disappears for more than 50 frames in the middle of the sequence. When the target reappears, TLD cannot resume tracking. In the face tracking adaptation, the face detector helps to reacquire the face and reinitilize TLD. In the middle row, the subject performs many rapid movements. Struck loses the target and fixates on the background. In the face tracking adaptation of Struck, face detection is performed periodically. This helps to correct the mistake and reacquire the true target. In the bottom row, the MIL tracker loses the target because of drastic lighting changes. The face tracking adaptation of MIL is able to track the target to the end of the sequence.

This face adaptation framework makes it easy to combine powerful state-of-the-art

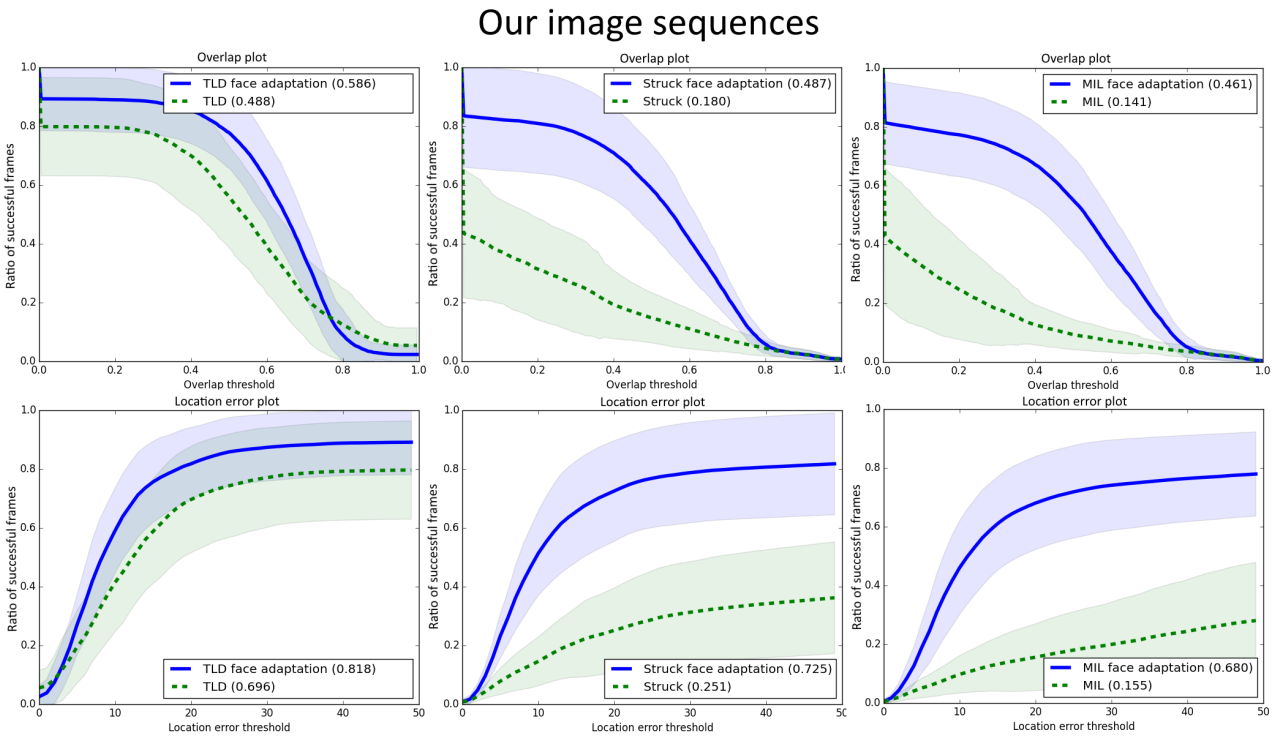

Figure 4.16: Performance evaluation on my image sequences. The overlap success plots for the three trackers are shown in the top row and the location precision plots are shown in the bottom row. Each line in a plot shows the mean performance over the image sequences and the shading shows one standard deviation.

trackers with a robust face re-detection system. I compared public datsets for face tracking with our dataset and showed that public datasets are not representative. I demonstrated using public datasets and my own dataset that this method improves tracking accuracy considerably. Potentially this method can be used for tracking objects other than faces if a corresponding detector is available. The Viola Jones approach, while usually used for faces, can be used for other objects if given appropriate training data.

It is interesting to note that in Figure 4.15 and Figure 4.16, the original TLD method without face detection has a higher success rate for overlaps more than 0.8 and location error less than 5 pixels. This can be explained by the fact that the ground truth face bounding boxes usually have elongated shapes to fit closely to the face shape. TLD retains the shape of the initialization bounding box to the end of a video. If in

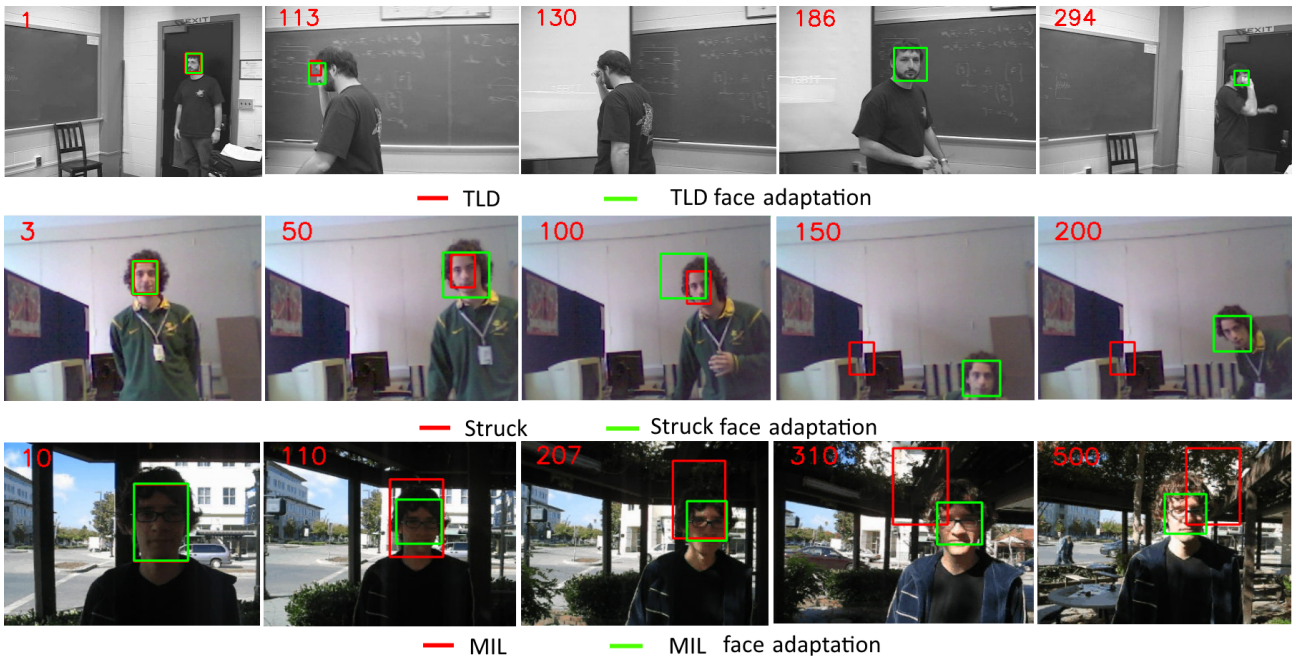

Figure 4.17: Examples of tracking results highlighting how face tracking adaptation helps to recover tracking after occlusions or rapid movements. The red boxes show the original trackers' performance and the green boxes show the face tracking adaptation performance.

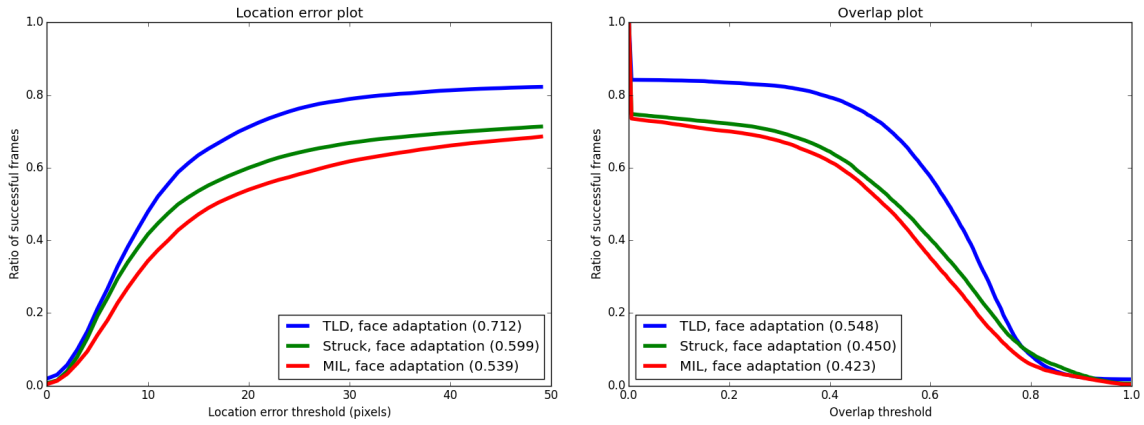

Figure 4.18: The comparison of TLD, Struck and MIL, adapted for tracking faces.

the beginning it is initialised by an elongated rectangle (the shape of a face bounding box) and the tracking is successful, then it will be a rectangle of the same proportions at the end of the sequence. At the same time the Viola Jones detector always outputs a square bounding box. If the tracker is reinitialized by the Viola Jones face detector, the

tracked target will have a square bounding box. This problem can be solved by using a face detector that outputs a rectangle closer to the real face shape or by correcting the face detector rectangles.

Figure 4.18 shows the comparison of TLD, Struck and MIL with the face adaptation. These results are shown for the joint dataset of our sequences and the public sequences. The face adapted TLD considerably outperforms the face adaptations of Struck and MIL.

4.3.5 Different modifications of TLD

The experiments in the previous subsection showed that the face adaptation of TLD outperforms the face adaptations of Struck and MIL. This subsection compares several different modifications of TLD:

- The original version of **TLD**.
- **FastTLD**, the fast version of TLD, described in Section 4.2.2.
- **Face adapted TLD**.
- The face adaptation of TLD with the TLD detector switched off. This version is basically **face adapted median flow**.
- **Face adapted FastTLD**. In this version the face detector is used to reinitialize the tracker. The TLD detector performs sparse detections, as described in section 4.2.2

The accuracy of the trackers is shown in Figure 4.19. This results are for the combined dataset of high resolution and low resolution sequences (including the public face tracking videos). The average frame rates are shown in Figure 4.23.

The following conclusions can be made:

- If accuracy is the most important then face adapted TLD should be used. It is possible to make it even more accurate by running the face detector on each frame.
- If speed is crucial, then it's preferable to use face adapted median flow. It also shows the second best accuracy.

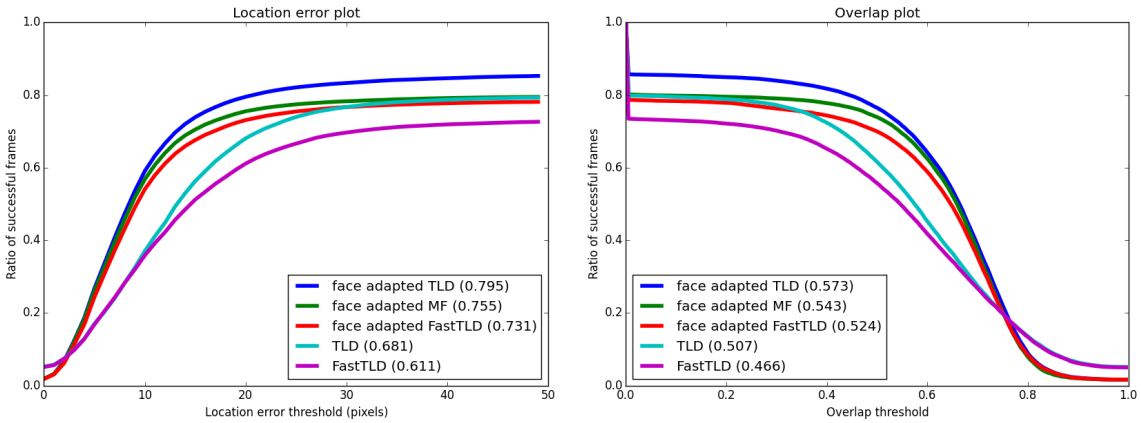

Figure 4.19: The performance of TLD, face adapted TLD, face adapted median flow.

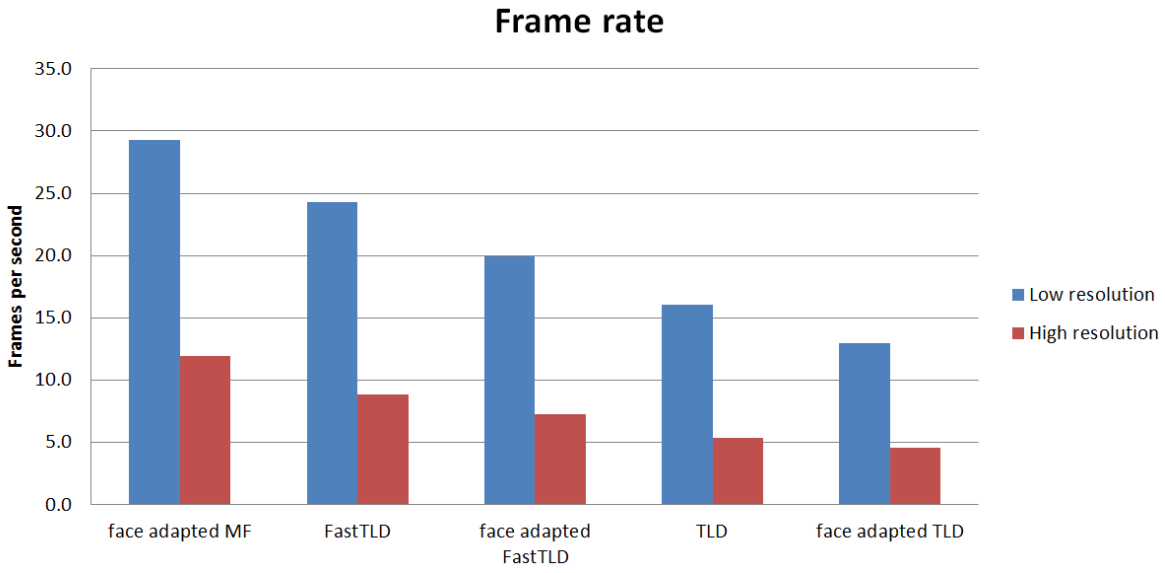

Figure 4.20: The frame rate summary for the trackers.

As I am trying to develop close to real time face tracking, I chose face adapted median flow for further investigation.

As it was discussed earlier, in Figure 4.19, the TLD methods without face detection (TLD and FastTLD) have higher success rates for overlaps more than 0.8 and location error less than 5 pixels. This is explained by the fact the ground truth face bounding boxes have elongated shape and at the same time the Viola Jones detector always outputs a square bounding box. If the tracker is reinitialized by the Viola Jones face detector, the tracked target will have a square bounding box (see Figure 4.21). This problem can be solved by using a face detector that outputs a rectangle closer to the

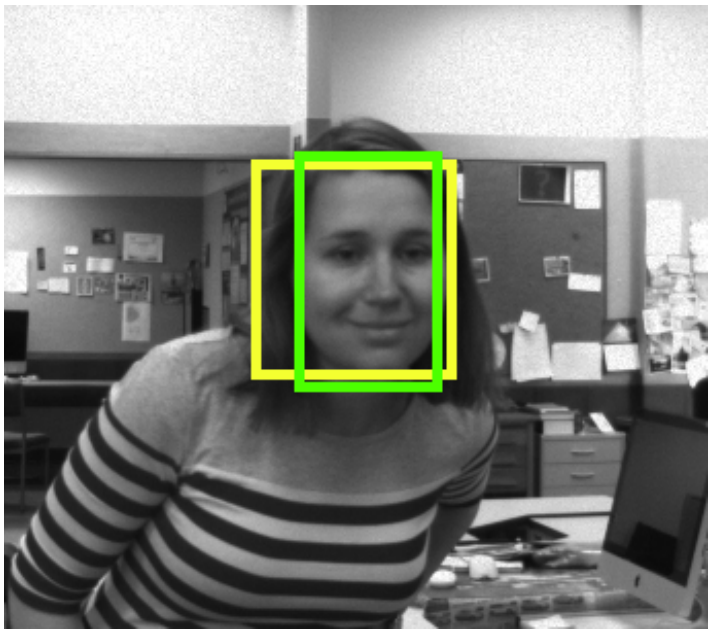

Figure 4.21: TLD retains the shape of the initial bounding box (green rectangle). If it is reinitialized by the face detector, the shape of the bounding box is changed to a square (yellow rectangle).

real face shape or by correcting the face detector rectangles.

4.4 Experimenting with parameters of face adapted median flow

The goal of this research is to develop a real time face tracking algorithm. I decided to carry on with face adapted median flow because face adapted median flow is the fastest modification of TLD and shows the second best accuracy, not much worse then the accuracy of face adapted TLD. This section describes experiments for finding the optimal parameters of face adapted median flow.

4.4.1 Frequency of running the face detector

The experiment described in this subsection estimates the optimal frequency of running the face detector. Any generic tracker if it can be restarted on demand, can be adapted for tracking faces using the method described above. However, running the face detector on every frame, especially on high resolution sequences, is very slow and

not necessary. The speed of the face detector depends on the detector implementation and the image size. The standard Viola Jones detector implemented in OpenCV scans a 1240×1024 image in 500 ms, and a 320×240 image in 50ms. If the tracker is supposed to run in real time, then a compromise between speed and accuracy must be found.

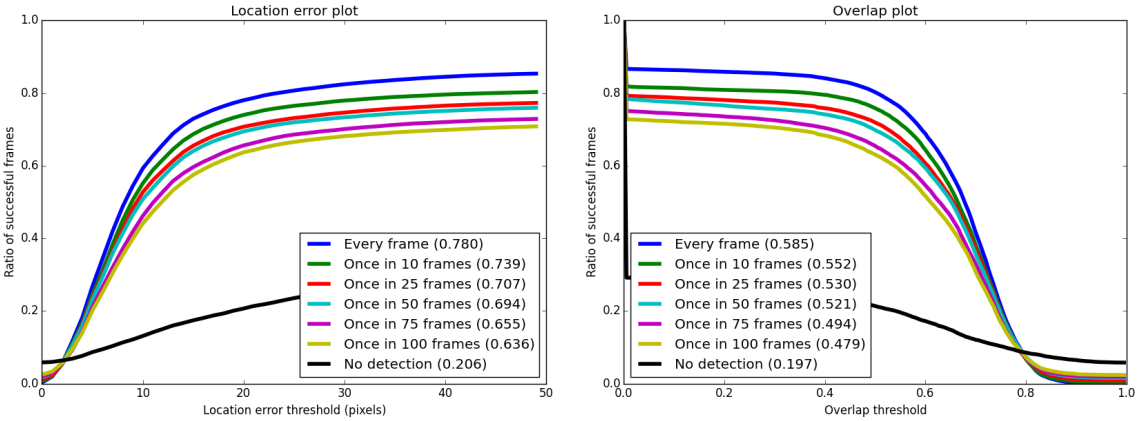

Figure 4.22: The performance of the median flow tracker adapted for tracking faces with different frequencies of running the face detector.

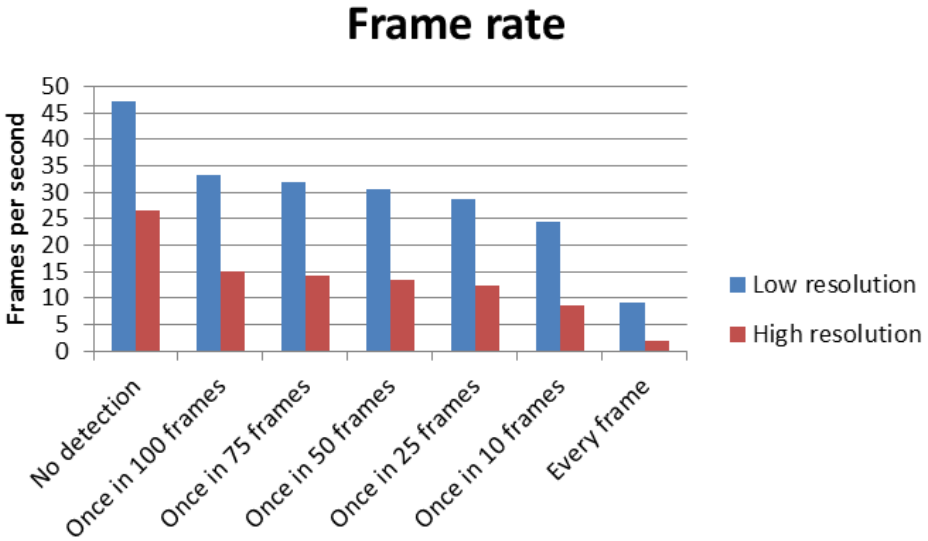

Figure 4.23: The frame rate summary for the median flow with different detection frequencies.

Figure 4.22 shows the performance of face adapted median flow with different detection frequencies. The accuracy of tracking decreases gradually with lowering the detection frequency. There is a big drop in accuracy when no detection is performed.

Median flow without face detection has higher success rates for overlaps more than 0.8 and location error less than 3 pixels. As before, this can be explained by the fact that if the tracker is reinitialized by the Viola Jones face detector, the tracked target will have a square bounding box. Otherwise, it will retain the shape of the initial bounding box. The ground truth bounding boxes also retain the shape of the initial bounding box. This results in a higher success rate for overlaps more than 0.8 for median flow without face detection

Figure 4.23 shows the corresponding frame rates. As explained in Section 3.4.3, the average frame rate is measured. The average frame rate can depend a lot on the tracking accuracy. On the frame rate plot, two extremes are visible: a very fast performance of the median flow tracker without detection and a very slow performance of the median flow tracker with detection on every frame. All the other trackers have similar frame rates. Is is important to note that when running the face detection once in certain number of frames, the overall tracking becomes jerky: it goes fast and smooth and then slows down on the detection stage. This problem can be solved by performing detection in a small part of the image on every frame (see Section 4.4.2).

4.4.2 Subwindow based detection

As mentioned in the previous section, running the face detector once in a certain number of frames makes the overall tracking jerky because tracking is much faster than detection, especially on large frames. To solve this problem, it is possible to use the same method as was used for speeding up TLD (see Section 4.2.2). Instead of performing detection on a full image once in a certain number of frames, the detection is performed each frame on two subwindows (Figure 4.24):

- The "following" subwindow around the previous target face position. The subwindow size depends on the previous target bounding box size. This subwindow is used to follow the target closely and correct the tracker if it start to divert from the target.

- Some subwindow in the background. This subwindow position changes each frame in a sliding window manner. This subwinow is used to monitor the background and find other people's faces and new confusing background patches to record them as negative examples.

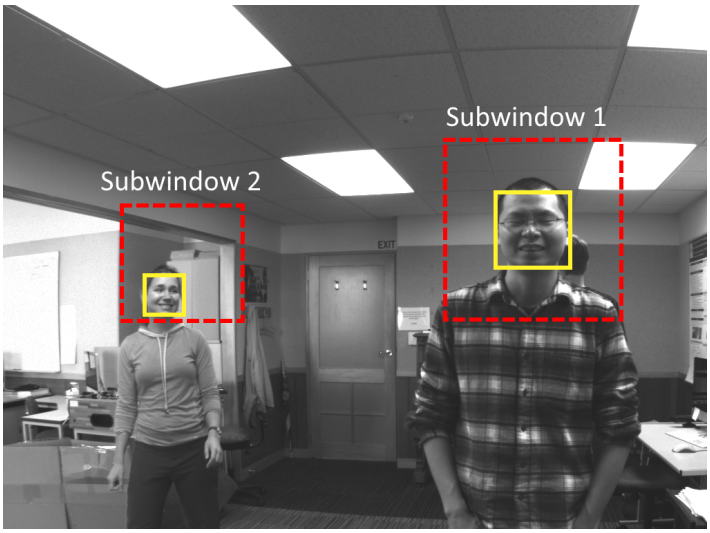

Figure 4.24: The face detection is performed in two subwindows. Subwindow 1 is a subwindow centred at the previous target position. Subwindow 2 is positioned in the background. It this example two faces are found. They are classified as either positive or negative examples in the next step.

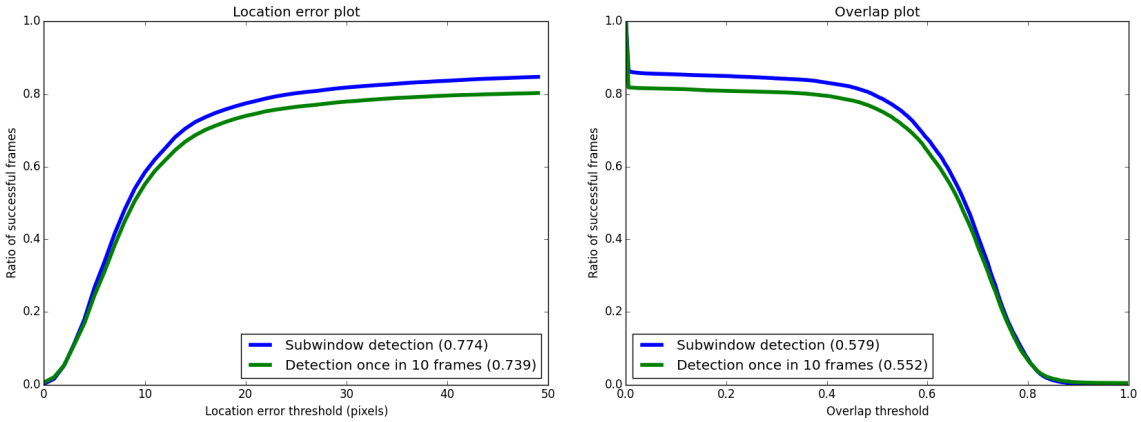

Figure 4.25: The performance of face adapted median flow with detection performed on a full frame once every 10 frames and face adapted median flow with subwindow based detection.

The faces found by the detector in these subwindows are classified as positive or negative examples according to the data structural constraints. Usually the face found in the subwindow around the previous head position is the true positive example of the target face. This example is added to the collection of positive examples. The faces found in the background subwindow are usually false positive examples (other

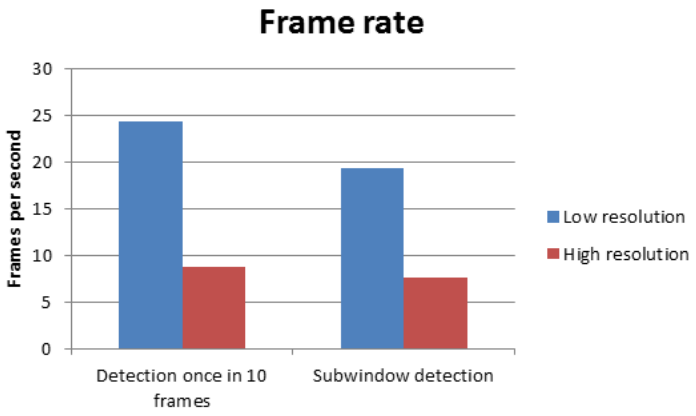

Figure 4.26: The frame rates of face adapted median flow with detection performed on a full frame once every 10 frames and face adapted median flow with subwindow based detection.

people's faces or face-like patches in the background). They are added to the collection of negative examples. The background subwindow position changes every frame in the sliding window manner. However, the full frame detection is performed if the target is considered to be lost in the previous frame.

In this experiment, the periodic detection is performed once in 10 frames. In the subwindow detection, the background subwindow covers 11% of the image area. The whole image area is covered in 9 frames. The results are shown in Figure 4.25. The two methods have similar accuracy. As shown in Figure 4.26, the frame rate of the subwindow detection is lower. However, it is more important that this methods provides smooth performance, which can be crucial in real time applications.

4.4.3 Different sliding window methods

As discussed in the previous section, the sliding window approach across several frames is used for positioning a detection subwindow. This travelling subwindow is used for examining the background in a search for new negative examples. A scan-line order is used because it doesn't matter where exactly a new negative example is found. Such examples can be the faces of other people entering the camera viewing zone. If the camera moves and the background changes, there might be new face-like patches in the background. It is important to detect these new face-like patches as soon as possible and save them to the collection of negative examples.

In the field of object localization with bounding boxes, sliding window approaches

have been the method of choice for many years (Castrillón *et al.*, 2010). Usually such localization methods rely on evaluating a quality function, e.g. a classifier score, over many rectangular subregions of the image and taking its maximum as the object's location. Because the number of rectangles of all sizes in an $n \times n$ image is of the order n^4, this maximization usually cannot be done exhaustively. Instead, several heuristics have been proposed to speed up the search. Typically, these consist of reducing the number of necessary function evaluations by searching only over a coarse grid. The coarser the grid is, the more efficient, but the higher the chance of missing the object of interest. Another approach, which can be used in combination with a coarse grid of subwindows, is to use ground plane constraints of the scene to skip irrelevant subwindows and evaluate only those patches that correspond to geometrically valid object detections (Sudowe and Leibe, 2011). Using ground plane constraints is a promising method. However because of limited time, I left it for future work.

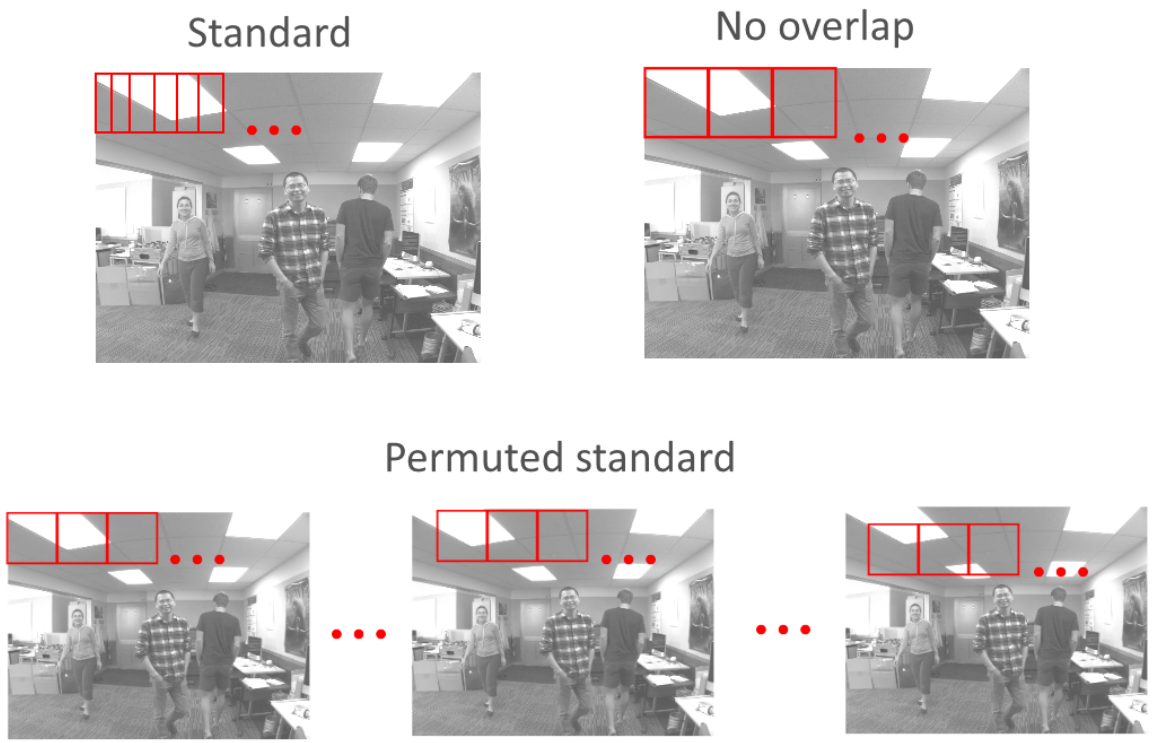

Figure 4.27: The illustration of the different sliding window methods.

I compared the standard sliding window method and my two simple modifications of it (see Figure 4.27):

- **The standard method**. Running the sliding window with a 0.2 overlap. The

scanning subwinow shape is a square and its height is 0.4 of the frame height. It takes 108 frames to cover the whole image. The downside of this method is that it takes so long to cover the background. Some of the important negative examples in the background can be missed while the scanning window is in the other part of the background.

- **No overlap**. Running the sliding window back to back with no overlap. Compared to the previous method, it takes only 9 frames to cover the whole image. The downside side is that the object of interest can be missed because it falls on the ridge between the two subwindows.
- **Permuted standard method**. Running the sliding window back to back with no overlap, but after running the full image area, starting over with a 0.2 shift. This method has advantages of the both previous methods and eliminates the downsides. It covers the whole image in 9 frames. It performs the same exhaustive work as the standard method in 108 frames.

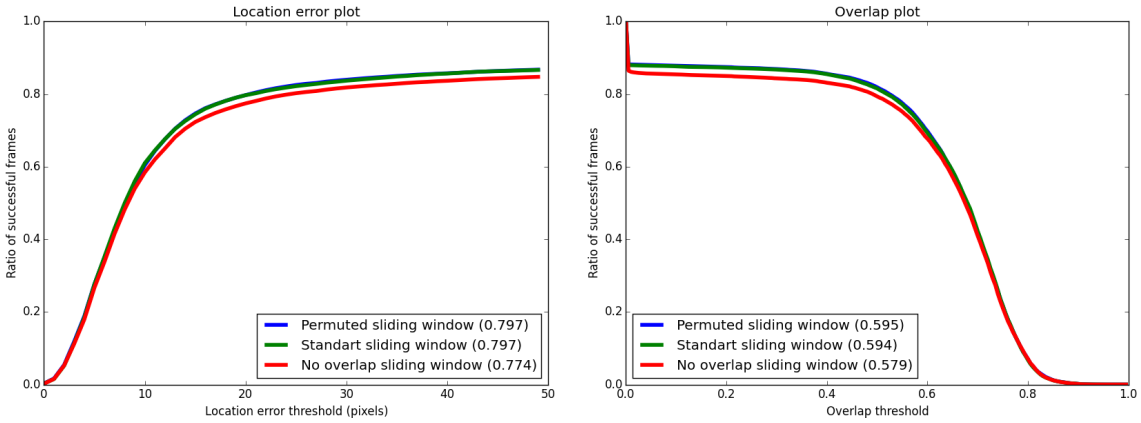

Figure 4.28: The performance of face adapted median flow with different sliding window methods

The scanning subwindow width and height are the same for all the three methods (0.4 of the image height). The results of the experiment are shown in Figure 4.28 and Figure 4.29. All three methods showed similar accuracy and speed. For the following experiments, I used the permuted standard method.

4.4.4 The "following" subwindow size

The performance of face adapted median flow with different "following" subwindow sizes was tested experimentally. Three sizes were tested: 3x3, 2x2, 1.5x1.5 of the

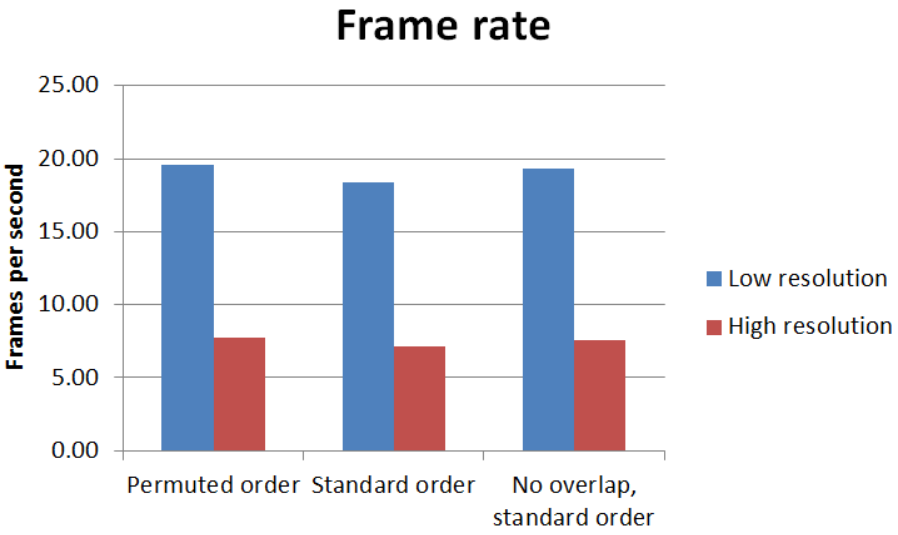

Figure 4.29: The frame rate of face adapted median flow with different sliding window methods.

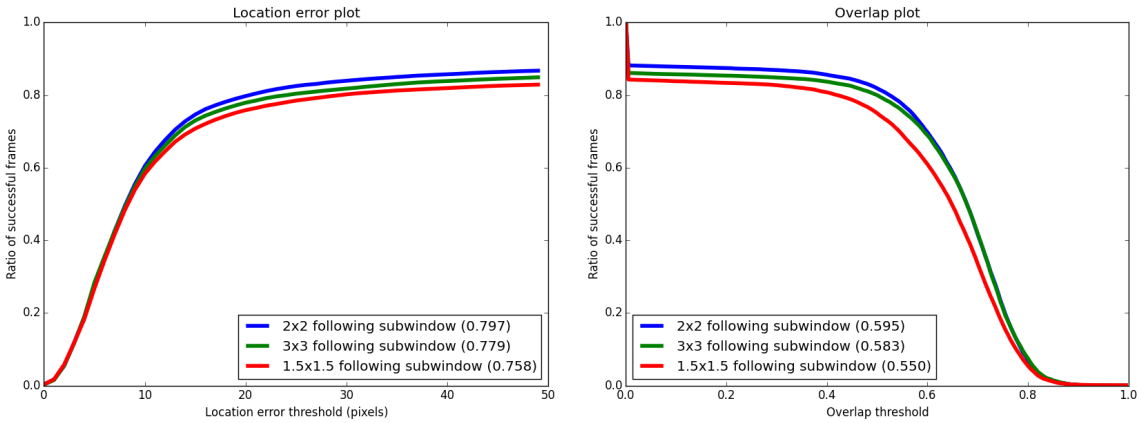

Figure 4.30: The performance of face adapted median flow with different sizes of the "following" subwindow.

previous target bounding box size. The background subwindow size (0.4 of the frame height) and shift (0.2 of the background subwindow size) were fixed in the experiment.

The results of the experiment are shown in Figure 4.30 and Figure 4.31. The accuracy is almost the same for all the three sizes, but the frame rate is slightly lower for the larger subwindow. The 2x2 size is used for the following experiments.

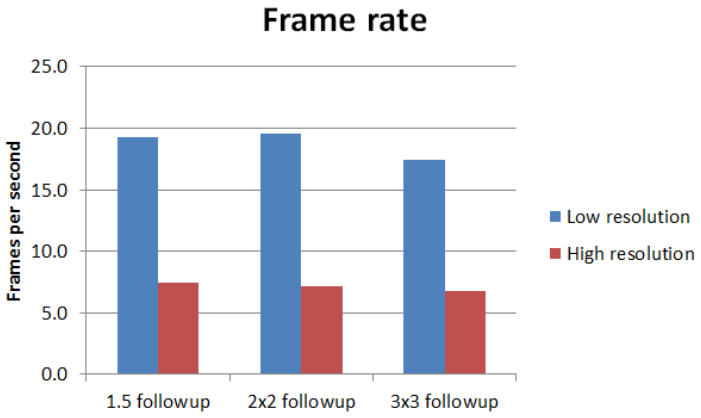

Figure 4.31: The frame rate of face adapted median flow with different sizes of the "following" subwindow.

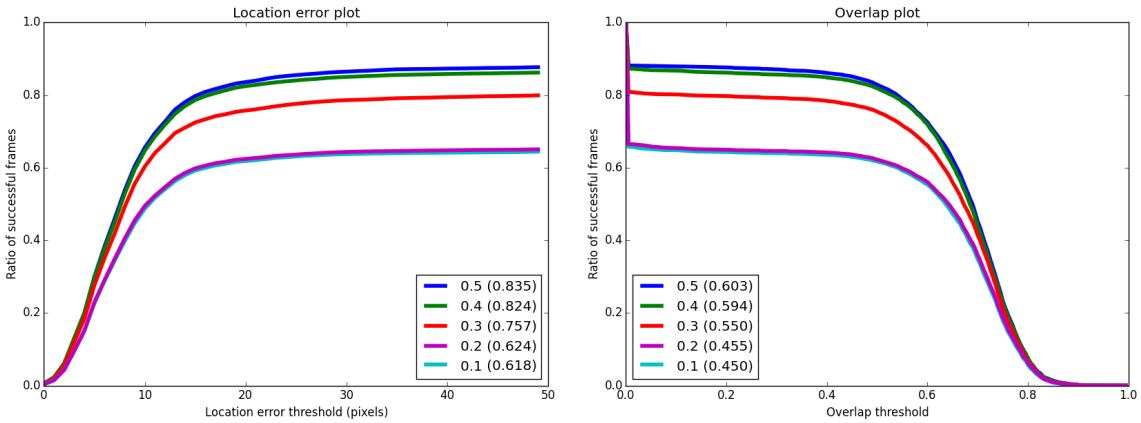

Figure 4.32: The accuracy of face adapted median flow with different sizes of the sliding window: 0.1, 0.2, 0.3, 0.4, and 0.5 of the frame height.

4.4.5 Sliding window size

This experiment was performed to check how the sliding window size affects speed and accuracy. Five sizes were checked: 0.1, 0.2, 0.3, 0.4, and 0.5 of the frame height. The accuracy is shown in Figure 4.32. The frame rates are shown in Figure 4.33. As expected, for larger sliding windows, the frame rate is lower. Note that the 0.1 sliding window is slower than the 0.2 sliding window in low resolution sequences. This happens because the accuracy of the 0.1 sliding window is low and more restarts are needed and this slows down the tracker.

The 0.4 window was used for the following experiments.

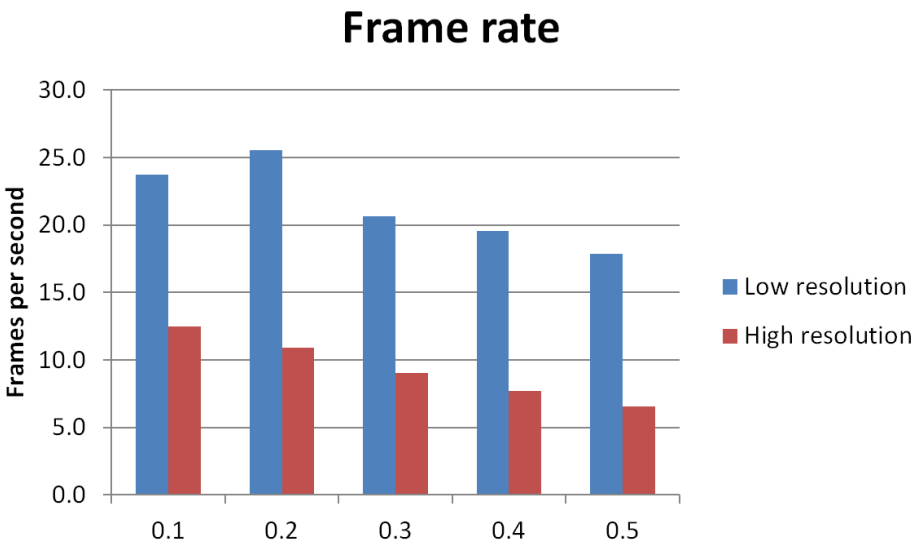

Figure 4.33: The frame rate of face adapted median flow with different sizes of the sliding window.

4.4.6 Average face size and Haar cascade limits

Another way to speed up face adapted median flow is to adjust the Haar cascade limits. The Haar cascade face detection takes a lot of time. I use the OpenCV implementation of the face detector, which is parallelized and highly optimised but still quite slow. The detector divides the searched window into subwindows of different sizes and checks each subwindow separately. It is possible to speed up the detector by reducing the number of subwindows that it has to check. The number of subwindows can be reduced by putting limits on the smallest face possible. It is not desirable to put the upper limit because the face can occupy the whole frame if the target person comes close to the camera. Also the lower limit cuts out many more subwindows than the upper limit.

Before deciding on the lower limit of the Haar cascade, I estimated the average face size in relation to the frame size in the high resolution videos in our dataset. As the face height is larger than its width, the average height was estimated. The results are shown in Figure 4.34.

For the videos with resolution of 1280×1024 pixels, 90% of faces have height between 0.03 and 0.23 of the frame height or between 51 and 143 pixels.

In the following experiment, two limits were compared: 5 x 5 pixels and 30 x 30 pixels. The results are shown in Figure 4.35. The 30 x 30 limit performs slightly worse. The 30x30 limit is too large and misses some of the small faces. Using the 30 x 30

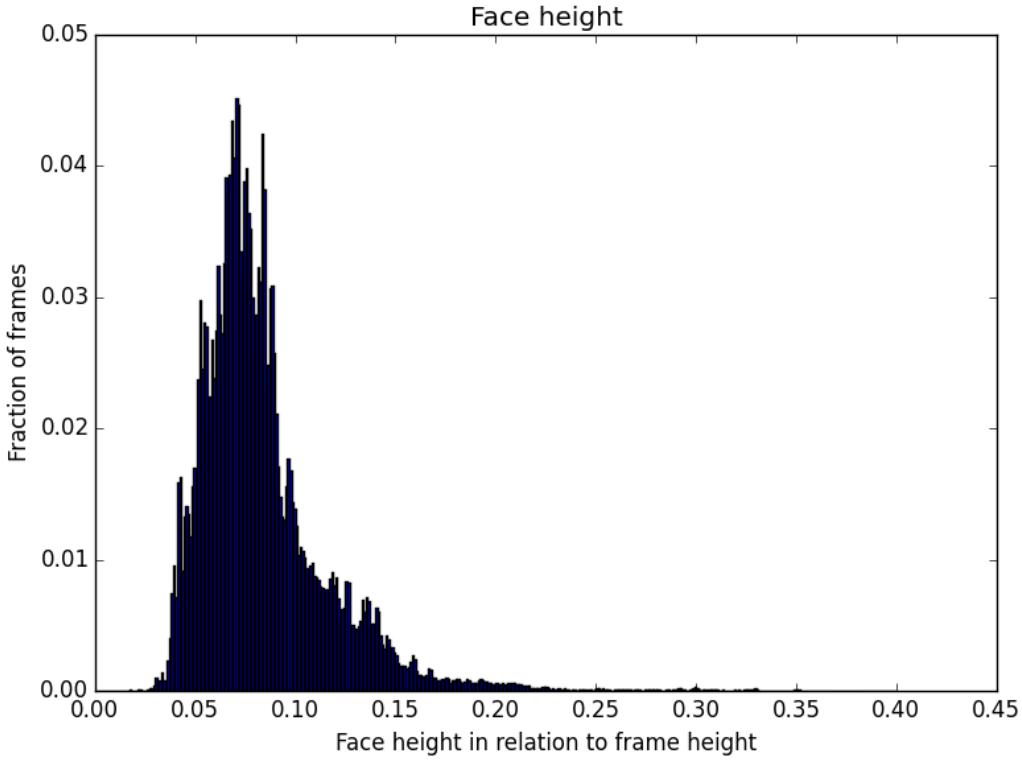

Figure 4.34: The face height distribution in the high resolution videos in our dataset.

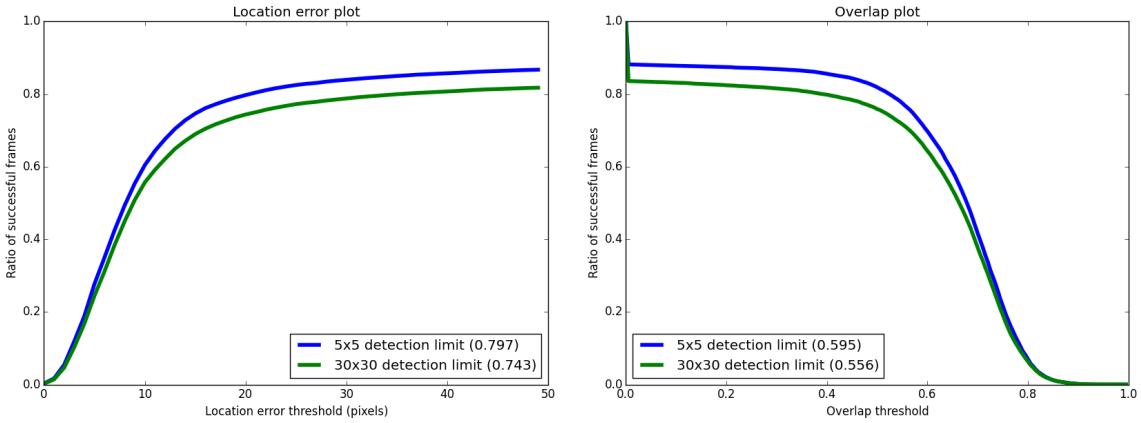

Figure 4.35: The performance of different detection limits in the high resolution videos.

limits helps to speed up the tracker performance (see Figure 4.36). It was decided to use the 5 x 5 limit for all videos in the following experiments.

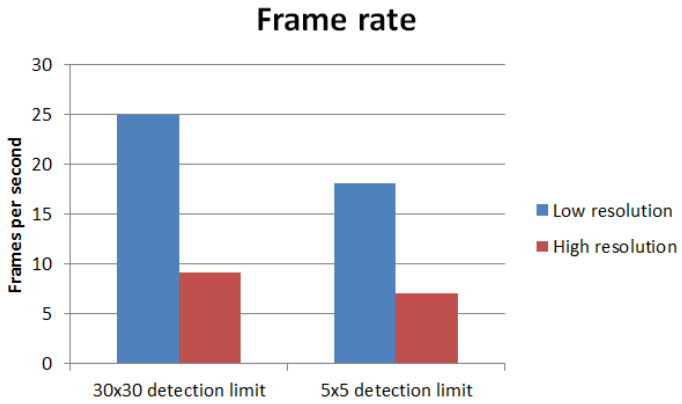

Figure 4.36: The frame rate for different detection limits.

4.4.7 Final comparison: TLD modifications and context tracker

The context tracker showed the best results on the public dataset and on our dataset in the experiment described in Section 4.1. To conclude this chapter, I compared the original TLD method, the context tracker, face adapted TLD and face adapted median flow. The results are shown in Figure 4.37 and Figure 4.38. Face adapted median flow is the fastest method and shows second best accuracy results, very close to the best result, shown by face adapted TLD. Developing this robust 2D face tracking algorithm is one of the main contributions of this thesis.

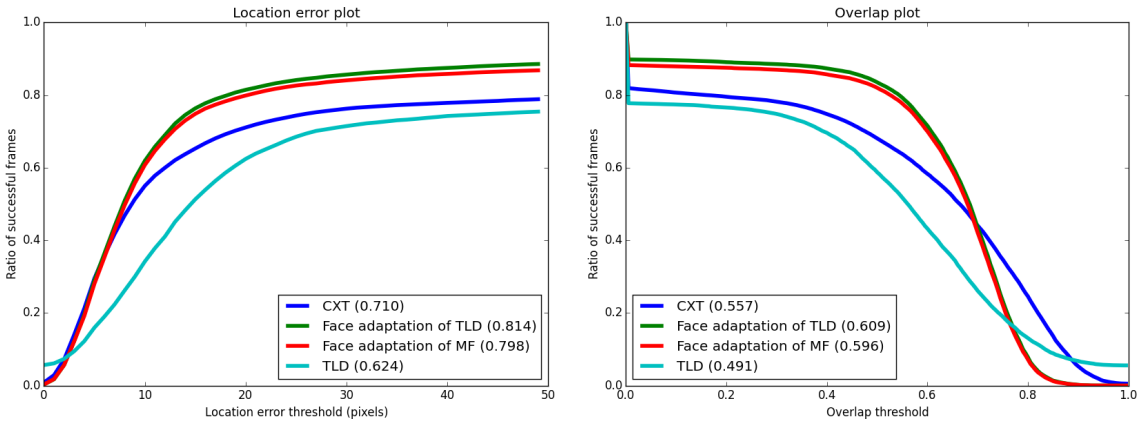

Figure 4.37: The final comparison of face adapted median flow, TLD, face adapted TLD and CXT.

Figure 4.37 is an example when the location error plot and overlap plot show slightly different results. In the location error plot, the context tracker, face adapted median flow and face adapted TLD show very similar success rates for location errors less than

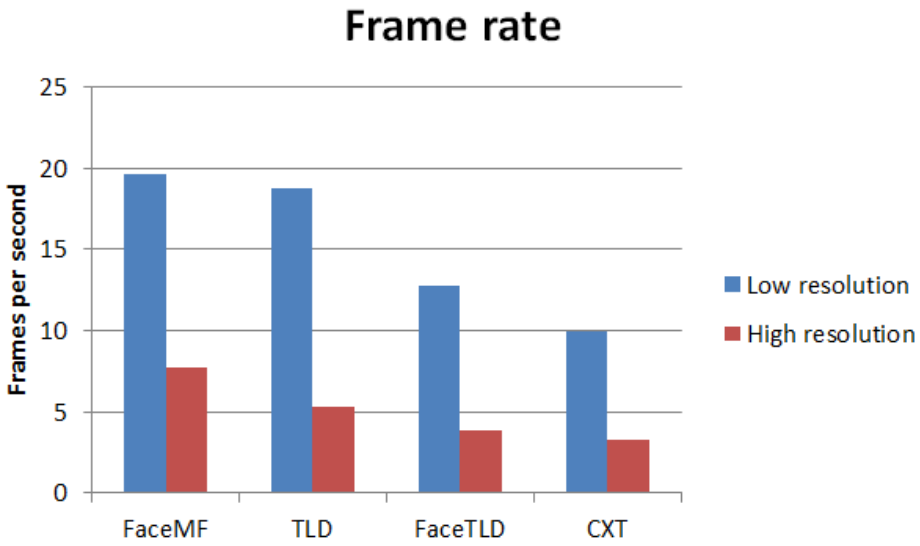

Figure 4.38: The frame rate comparison for face adapted median flow, TLD, face adapted TLD and CXT.

7 pixels. At the same time in the overlap plot, the context tracker has higher success rates for the overlaps higher than 0.7. This can be explained by the fact that the context tracker retains the shape of the initial rectangular face bounding box, while face adapted median flow and face adapted TLD get a square bounding box when reinitialized by the Viola Jones face detector. A square bounding may not affect the location error, but definitely affects the overlap with the ground truth face box, as most ground truth face bounding boxes have an elongated rectangular shape. For this reason in Figure 4.37, TLD shows higher success rates for overlaps higher than 0.9.

4.5 Conclusion

This chapter was concerned with developing a 2D long-term tracking algorithm for tracking faces in whole body view video sequences. As initial step towards this goal, I explored the existing publicly available generic trackers on face tracking datasets. TLD and the context tracker showed the best performance. Then a novel method of adapting generic trackers for tracking faces was developed and evaluated. The experiments showed that face adaptation of generic tracker improves their accuracy considerably on the face tracking datasets. For real time applications, face adapted median flow shows the most suitable combination of speed and accuracy. On low

resolution images (640×512) it can run at around 19 frames per second which is close to real time. Also, I performed a set of experiments to find the best parameters for face adapted median flow. The experiments show that performing detection in the sliding window manner across several frames is better than running it periodically on a full frame. The parameters such as the cascade detector limits and the detection subwindow size and position do not play an important role.

In the next chapter we will look into developing a 3D face tracking algorithm on the base of face adapted median flow.

Chapter 5

3D Face Tracking

This chapter investigates building a 3D face tracker based on the 2D face adapted median flow tracker described in the previous chapter. The straightforward way to do that is to run two instances of 2D face adapted median flow and then triangulate the results to find the target's 3D position. However, the stereo feed coming from the two cameras contains useful information that can be employed to facilitate 3D tracking. Therefore the goal of this chapter is to find ways of using the stereo information in the most effective manner.

The chapter is structured as follows. Section 5.1 focuses on converting a 2D tracking algorithm into 3D algorithm. 3D face adapted median flow is experimentally compared to 3D TLD. Section 5.2 investigates methods of using stereo information to improve tracking accuracy. A novel method of checking the consistency of the target size and 3D position is developed. This method helps to recognize tracking failures early and restart the tracking. Section 5.3 studies ways of converting the 2D Lucas-Kanade tracker into 3D to speed up the final 3D face adapted median flow tracker.

5.1 Converting 2D tracking into 3D tracking by triangulation

In Chapter 4 I introduced face adapted median flow. The straightforward way to convert this fast and accurate 2D tracker into 3D is to run two independent instances of the tracker. The tracking results (consisting of a bounding box around the subject's face) are then triangulated to find the 3D head position. The block scheme of this 3D tracker is shown in Figure 5.1. Any other 2D tracker can be converted into 3D the same way. I converted 2D TLD into 3D TLD according to this scheme.

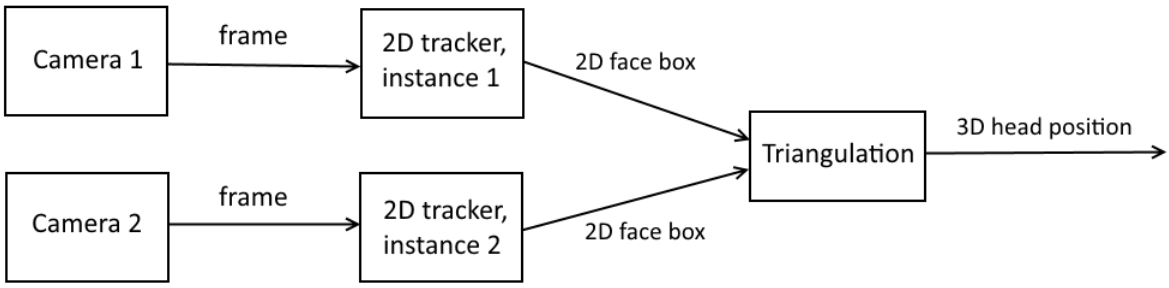

Figure 5.1: The block scheme of a simple 3D tracker composed out of two independent instances of a 2D tracker. This way any 2D tracker can be converted into 3D.

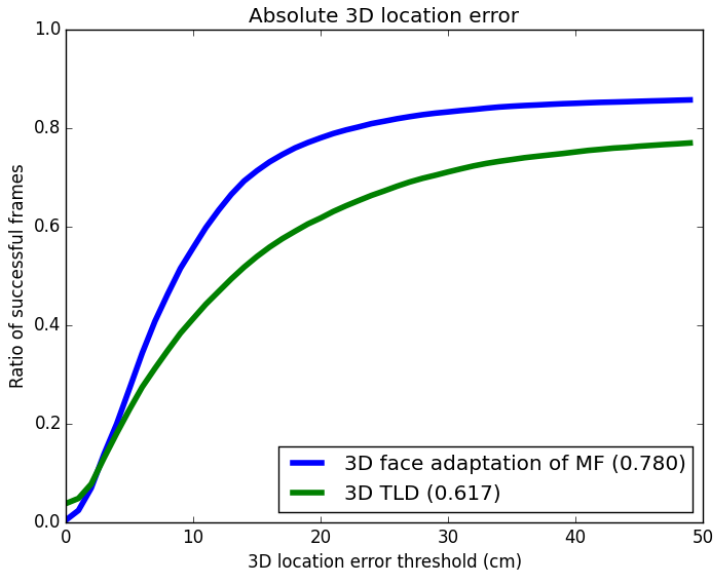

Figure 5.2: The location error plot for 3D face adapted median flow and 3D TLD. The success ratio at the threshold of 20 cm is given in brackets after the trackers names.

As there are no publicly available 3D visual trackers to compare to my 3D trackers, I compared 3D face adapted median flow to 3D TLD that I developed according to the scheme in Figure 5.1. The performance comparison of the two 3D trackers is shown in Figure 5.2 (for an explanation of the 3D tracking performance metric see Section 3.4.2). This experiment and all the following experiments in this chapter were performed on my high resolution and low resolution stereo datasets (see Section 3.1 for the datasets

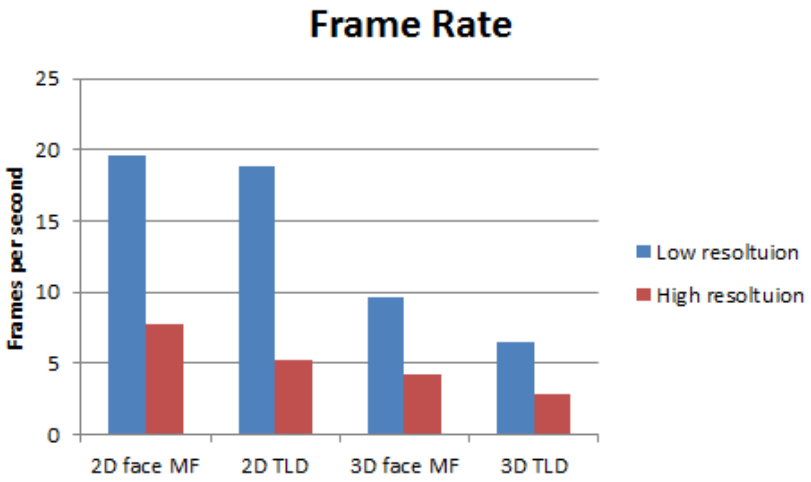

Figure 5.3: The frame rate plot for the 2D and 3D face adapted median flow and 2D and 3D TLD.

description). As 2D face adapted median flow outperforms 2D TLD, there is no surprise that 3D face adapted median flow outperforms 3D TLD. The frame rates are shown in Figure 5.3. 3D face adapted median flow is considerably faster than 3D TLD. The frame rate for 2D face adapted median flow and 2D TLD are also shown in this plot to illustrate that they are approximately two times as fast as the 3D versions of these trackers.

On the low resolution images, 3D face adapted median flow runs on around 9 fps. At high resolution, the speed is around 4 fps. This is considerably slower than the required real time performance (30 frames per second). The slowest part of face adapted median flow is the Viola Jones face detector. According to Pulli *et al.* (2012), the GPU implementation of the Viola Jones detector is on average 6 times faster than the CPU implementation. Wai *et al.* (2015) managed to achieve the 22 times speed up of the detector on 640×480 images using the NVidia K40 card. These results indicate that it is possible to make face adapted median flow faster and possibly close to real time.

5.2 Using stereo information

Building a 3D tracker out of two independent instances of a 2D tracker is not efficient as the 2D trackers do not make use of stereo information. By stereo information I mean all the mutual dependencies between the video feeds of the two cameras overlooking the

same scene. For example, if an object in the shared field of view moves, the direction of its projections movements in the two cameras feeds must agree. Using stereo information can improve tracking performance and speed up the tracking. For review of multiple ways of using stereo information see Section 2.1. This section discusses using stereo information to assist tracking accuracy and speed. I first examine the use of size and 3D position consistency in the stereo camera feeds to detect tracking failures. Then I explore methods for sharing information (examples of the target appearance) between the trackers with the aim of increasing the accuracy and speed of the tracking.

Figure 5.4: The illustration of the epipolar constraint. The face is denoted by the bounding box in the left image. In the right image the epipolar line corresponding to the center of the bounding box is shown.

5.2.1 Checking size and 3D position

The 3D tracker consisting of two independent 2D trackers does not use any stereo information. Therefore, it is important to check that the two 2D tracking results give a consistent 3D result. This can be done by using the epipolar constraint and the target size. The epipolar constraint means that for a point observed in one image the same point must be observed in the other image on a known epipolar line. An illustration of an epipolar line is shown in Figure 5.4. Another constraint is given by a fixed object size. In our case the head size is known. It is possible to use an average head size or to compute the tracked head size at the initialization step. In the following procedure, we can use these two constraints to check if the tracking results in both views give a

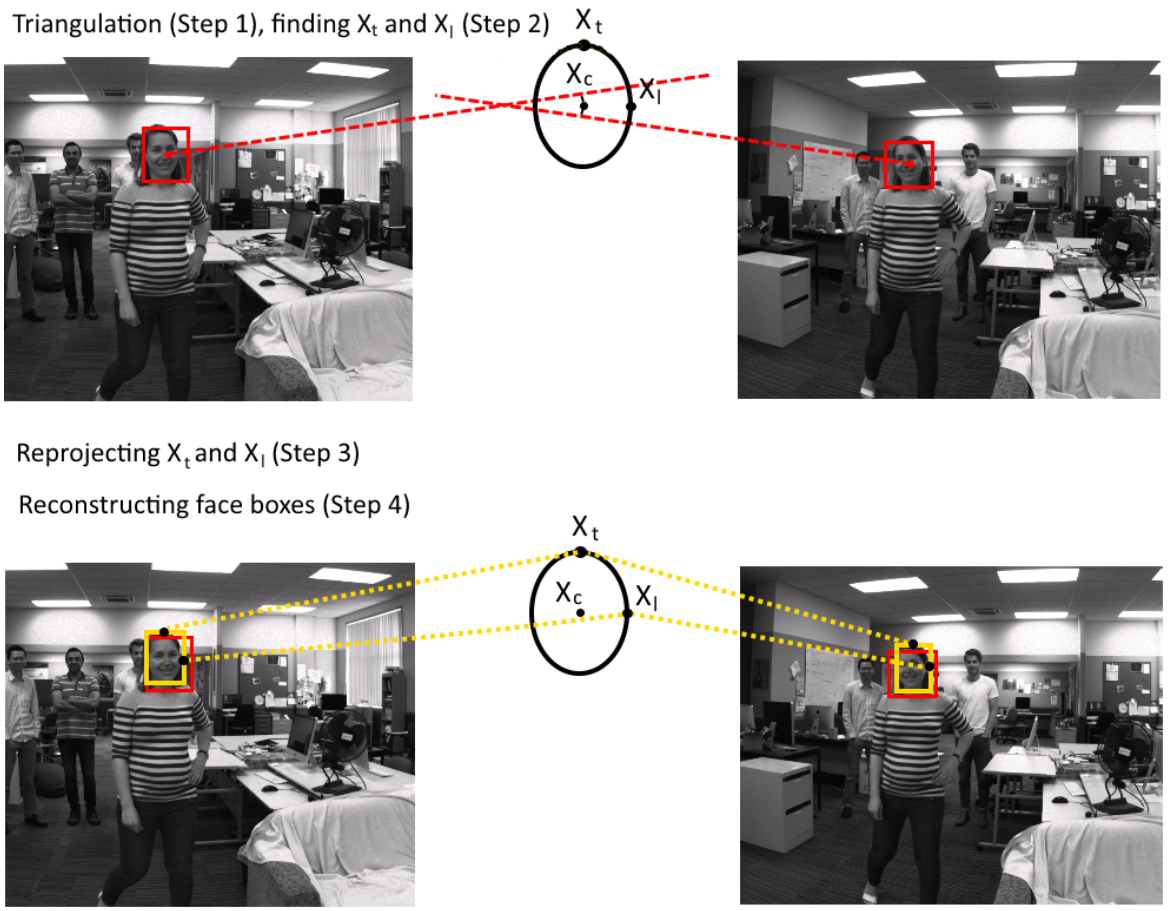

Figure 5.5: Checking the consistency of the 3D result.

meaningful 3D head position. This method is one of the contributions of this thesis. The procedure is illustrated in Figure 5.5 and can be summarised as follows:

1. The two 2D tracking results (the centres of the face bounding boxes) are triangulated to find the 3D position of the head centre X_c.
2. The 3D positions of the top of the head X_t and the left middle point X_l are computed.
3. These two 3D points X_t and X_l are reprojected back into the left and right images.
4. The face boxes are reconstructed from the two reprojected points.
5. The overlap between the tracking boxes and the reconstructed boxes is computed.

Ideally, these two reconstructed boxes should match the tracking boxes precisely, but this is never the case. Therefore, we assume that the tracking results are correct if the overlaps between the tracking boxes and the reconstructed boxes in both views are above a certain threshold. If the overlap is lower than the threshold in either view, then this indicates that one of the trackers started to drift away from the target, and the two trackers do not produce a meaningful 3D result.

In this case it is important to decide which of the two 2D trackers has failed. To do that, the confidence of both tracking results is computed. To compute the confidence of a face candidate, it is compared to the collections of positive and negative examples (for more details see Section 4.3.3). The result with the lower confidence is discarded and the face search is performed in that image.

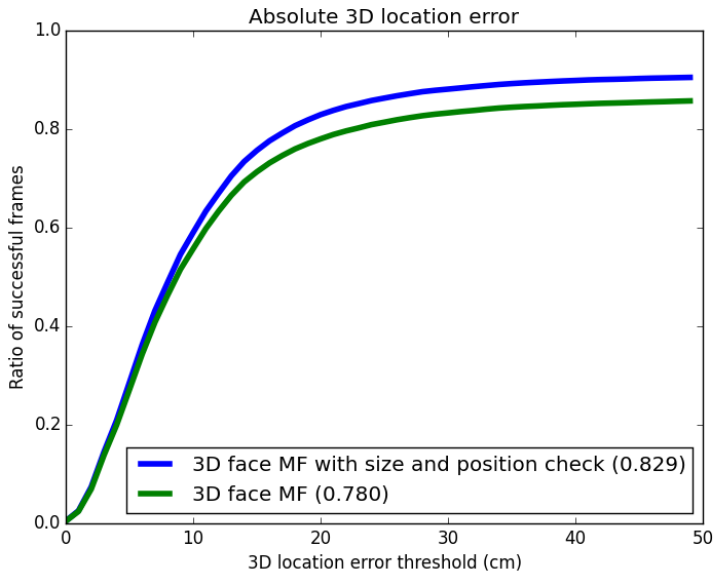

Figure 5.6: The location error plot for 3D face adapted median flow with the size and 3D position check and without it.

An experiment was performed to find out if the described size and 3D position check helps to detect failures earlier and correct them. I compared a simple 3D tracker composed out of two independent 2D face adapted median flow trackers and a 3D face adapted median flow tracker with the size and 3D position check performed on every pair of stereo frames. The results are shown in Figure 5.6. 3D face adapted median flow with the size and 3D position check does show better accuracy than 3D face adapted median flow without the check. The frame rate plot is not provided because the size

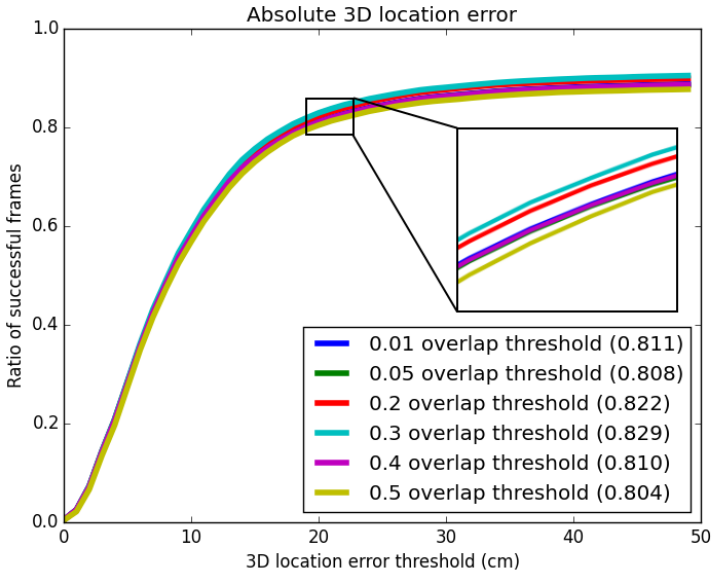

Figure 5.7: The location error plot for different overlap thresholds.

and 3D position check is a very fast operation and it does not affect the tracking speed.

I performed another experiment to find out which overlap threshold shows better results. If the threshold is too small, a significant violation of the epipolar constraint or the head size constraint may go undetected for a while. If the threshold is too high, the tracker can be restarted more often than needed because there is never 100% alignment between the tracking box and the reconstructed face box. The results of different overlap thresholds are shown in Figure 5.7. All thresholds show similar results, but the 0.3 threshold is slightly better.

5.2.2 Sharing information between the views

In our stereo setup, the two cameras have quite similar views of the scene. Therefore the two 2D trackers might benefit from sharing information between each other. As discussed in Section 4.3.3, each of the two face adapted median flow trackers maintain its own collections of true positive and false positive face examples. As the two cameras have similar views of the scene, the 2D trackers collect similar positive and negative examples. It is possible that the examples collected by one tracker can make the other tracker more accurate. In this experiment, I check if sharing the information about the face's appearance between the 2D trackers can improve the overall performance. The shared collections of positive and negative examples are illustrated in Figure 5.8.

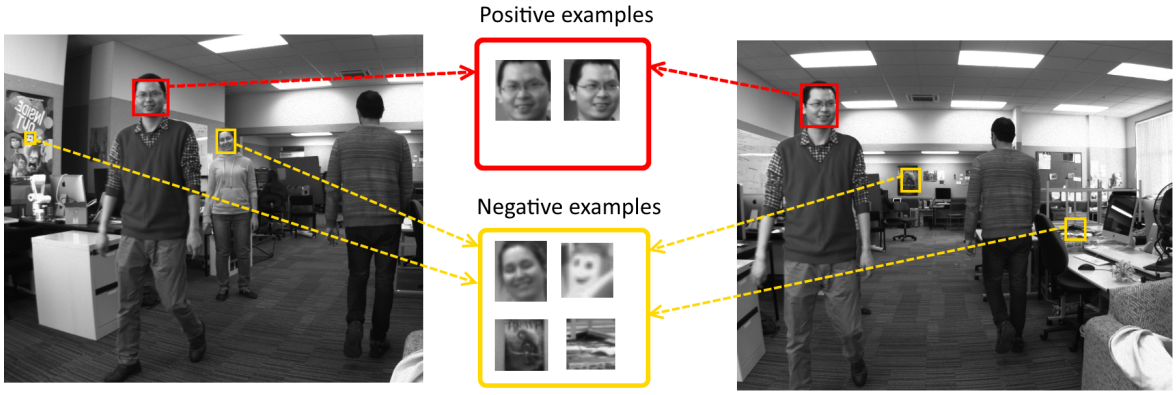

Figure 5.8: The collections of positive and negative examples are shared between the views.

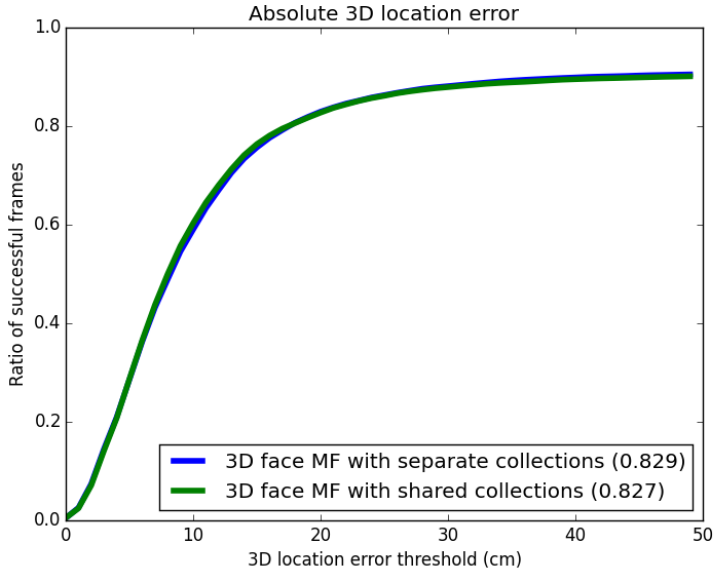

Figure 5.9: The location error plot for 3D face adapted median flow with shared face examples collections and with separate collections.

The accuracy of this approach is shown in Figure 5.9, and there is very little difference between the two 3D trackers. On our dataset, the trackers do not show any accuracy improvement when sharing information about what the face looks like.

In this experiment, an interesting observation can be made about the collections sizes. The collections grow when new examples are added to them. As explained in Section 4.3.3, every frame the structural constraints create new positive and negative examples. However, these new examples are added to the collections if only they are

different enough from the examples that are already collected. For more details on how it is performed, see Section "P/N-learning" in Nebehay (2012). This experiment showed that when sharing the information between the views, the size of the shared collection of positive examples is approximately equal to the combined size of the two separate positive collections when the information is not shared. The same is true for the negative collections. This fact can lead to a conclusion that the examples in one view are not similar to the examples in the other view even though they look similar to a human eye. It does not really matter whether we put all positive examples into one bin or into two separate bins, one for each tracker. Each 2D face adapted tracker uses its own sets of positive and negative examples.

5.3 3D median flow tracker

One of the main components of our 2D face adapted median flow is the median flow tracker. The median flow tracker is the Lucas-Kanade tracker (Lucas and Kanade, 1981) with the forward-backward (FB) failure detection (Kalal *et al.*, 2010b). It is possible to convert it into a 3D median flow tracker. In this section I investigate and compare several 3D median flow trackers in an attempt to optimize and speed up the final 3D face adapted median flow tracker.

5.3.1 3D Lucas-Kanade tracking

The median flow tracker is the Lucas-Kanade tracker with the forward-backward (FB) failure estimation. The FB failure estimation will be described later in Section 5.3.3. The original 2D Lucas-Kanade method is described in Lucas and Kanade (1981). The pyramid implementation of this method can be found in Bouguet (2001). The 3D Lucas-Kanade tracking algorithm can be found in Harguess *et al.* (2011) and Xiao *et al.* (2002) and is described here.

In general, the 2D Lucas-Kanade method studies the spatial and temporal gradient of a local region of an image to find, where this region moved in the next frame. The 3D Lucas-Kanade method performs the same procedure, but the spatial and temporal gradient is computed for the two pairs of images that come from the two cameras. The mathematical description follows.

Let us assume that we have two stereo calibrated cameras: C and C'. The main coordinate system, in which we work, coincides with the coordinate system of camera

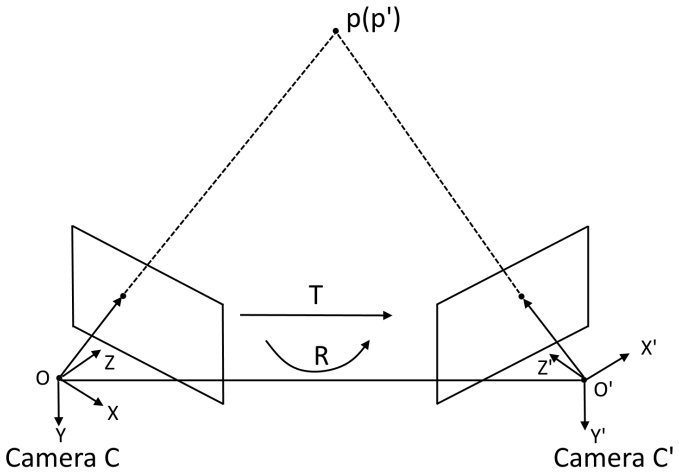

Figure 5.10: The 3D point $\mathbf{p}$ in the C system is projected to the 3D point $\mathbf{p}'$ in the C' coordinate system.

C. The 3D point $\mathbf{p}$ in the C system is projected to the 3D point $\mathbf{p}'$ in the C' coordinate system (see Figure 5.10):

$$\mathbf{p}' = \mathbf{R}\mathbf{p} + \mathbf{t} \tag{5.1}$$

where $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation vector between the coordinate systems of C and C'. Let I_t be an image from camera C and I'_t be an image from camera C' at moment t. Let I_{t+1} and I'_{t+1} be another pair of images taken in the next moment. Consider a 3D point in space $\mathbf{m} = [m_x, m_y, m_z]^T$. The goal of tracking is to find the location $\mathbf{n} = \mathbf{m} + \mathbf{d} = [m_x + d_x, m_y + d_y, m_z + d_z]$, such that these points projections in camera C images are similar $I_t(\mathbf{m}) \approx I_{t+1}(\mathbf{n})$ and the same is true for the pair of images from camera C': $I'_t(\mathbf{m}') \approx I'_{t+1}(\mathbf{n}')$. To find $\mathbf{d}$, we minimize the residual function E defined as follows:

$$E(\mathbf{d}) = E_1(\mathbf{d}) + E_2(\mathbf{d}') \tag{5.2}$$

$$E(\mathbf{d}) = \sum_{\Omega}(I_t(\mathbf{m}) - I_{t+1}(\mathbf{m} + \mathbf{d}))^2 + \sum_{\Omega}(I'_t(\mathbf{m}') - I'_{t+1}(\mathbf{m}' + \mathbf{d}'))^2, \tag{5.3}$$

where Ω is a neighbourhood of pixels around the 2D projection of the point $\mathbf{m}$. To minimize E, we set its derivative with respect to $\mathbf{d}$ to zero:

$$\frac{\partial E}{\partial \mathbf{d}} = \frac{\partial E_1}{\partial \mathbf{d}} + \frac{\partial E_2}{\partial \mathbf{d}'}\frac{\partial \mathbf{d}'}{\partial \mathbf{d}} = 0 \tag{5.4}$$

Knowing that $\mathbf{d}' = \mathbf{R}\mathbf{d}$, we can rewrite this as:

$$\frac{\partial E_1}{\partial \mathbf{d}} + \frac{\partial E_2}{\partial \mathbf{d}'}\mathbf{R} = 0 \tag{5.5}$$

For camera C we get:

$$\frac{\partial E_1}{\partial \mathbf{d}} = -2\sum_{\Omega}(I_t(\mathbf{m}) - I_{t+1}(\mathbf{m} + \mathbf{d}))\begin{bmatrix}\frac{\partial I_{t+1}}{\partial d_x} & \frac{\partial I_{t+1}}{\partial d_y} & \frac{\partial I_{t+1}}{\partial d_z}\end{bmatrix} \tag{5.6}$$

Let us now substitute $I_{t+1}(\mathbf{m}+\mathbf{d})$ by its first order Taylor expansion about the point $\mathbf{d} = [0 \quad 0 \quad 0]^T$:

$$\frac{\partial E_1}{\partial \mathbf{d}} = -2\sum_{\Omega}\left(I_t(\mathbf{m}) - I_{t+1}(\mathbf{m}) - \begin{bmatrix}\frac{\partial I_{t+1}}{\partial d_x} & \frac{\partial I_{t+1}}{\partial d_y} & \frac{\partial I_{t+1}}{\partial d_z}\end{bmatrix}\mathbf{d}\right)\begin{bmatrix}\frac{\partial I_{t+1}}{\partial d_x} & \frac{\partial I_{t+1}}{\partial d_y} & \frac{\partial I_{t+1}}{\partial d_z}\end{bmatrix} \tag{5.7}$$

The quantity $I(\mathbf{m}) - I_{t+1}(\mathbf{m})$ can be interpreted as the temporal image derivative at the point $\mathbf{m}$:

$$\delta I_t(\mathbf{m}) \doteq I(\mathbf{m}) - I_{t+1}(\mathbf{m}) \tag{5.8}$$

Suppose that the 3D point $\mathbf{m}$ is projected to some 2D point with pixel coordinates (u, v):

$$u = \frac{f_1 m_x}{m_z} + C_{1x}, \quad v = \frac{f_1 m_y}{m_z} + C_{1y}, \tag{5.9}$$

where f_1 is the camera C focal center, and (C_{1x}, C_{1y}) is the optical center. The matrix $\begin{bmatrix}\frac{\partial I_{t+1}}{\partial d_x} & \frac{\partial I_{t+1}}{\partial d_y} & \frac{\partial I_{t+1}}{\partial d_z}\end{bmatrix}$ is the image gradient vector with respect to the 3D displacement $\mathbf{d}$:

$$\nabla I = \begin{bmatrix}I_x \\ I_y \\ I_z\end{bmatrix} = \begin{bmatrix}\frac{\partial I_{t+1}}{\partial d_x} \\ \frac{\partial I_{t+1}}{\partial d_y} \\ \frac{\partial I_{t+1}}{\partial d_z}\end{bmatrix} = \begin{bmatrix}\frac{\partial I_{t+1}}{\partial u}\frac{\partial u}{\partial d_x} + \frac{\partial I_{t+1}}{\partial v}\frac{\partial v}{\partial d_x} \\ \frac{\partial I_{t+1}}{\partial u}\frac{\partial u}{\partial d_y} + \frac{\partial I_{t+1}}{\partial v}\frac{\partial v}{\partial d_y} \\ \frac{\partial I_{t+1}}{\partial u}\frac{\partial u}{\partial d_z} + \frac{\partial I_{t+1}}{\partial v}\frac{\partial v}{\partial d_z}\end{bmatrix} = \begin{bmatrix}\frac{\partial I_{t+1}}{\partial u}\frac{f_1}{m_z} + 0 \\ 0 + \frac{\partial I_{t+1}}{\partial v}\frac{f_1}{m_z} \\ -\frac{\partial I_{t+1}}{\partial u}\frac{m_x f_1}{m_z^2} - \frac{\partial I_{t+1}}{\partial v}\frac{m_y f_1}{m_z^2}\end{bmatrix} \tag{5.10}$$

The image derivatives $\frac{\partial I_{t+1}}{\partial u}$ and $\frac{\partial I_{t+1}}{\partial v}$ are computed using the Sharr operator. Following the new notation, equation 5.7 may be rewritten as:

$$\frac{1}{2}\frac{\partial E_1}{\partial \mathbf{d}} = \sum_{\Omega}(\nabla I^T \mathbf{d} - \delta I)\nabla I^T \tag{5.11}$$

For camera C' all the computations are the same as those for C, but they are performed in the C' coordinate system:

$$\frac{1}{2}\frac{\partial E_2}{\partial \mathbf{d}} = \frac{1}{2}\frac{\partial E_2}{\partial \mathbf{d}'}\mathbf{R} = \sum_{\Omega}(\nabla I'^T\mathbf{d}' - \delta I')\nabla I'^T\mathbf{R} = \sum_{\Omega}(\nabla I'^T\mathbf{R}\mathbf{d} - \delta I')\nabla I'^T\mathbf{R} \quad (5.12)$$

Denote:

$$\mathbf{G}_1 \doteq \sum_{\Omega}\nabla I^T\nabla I, \quad \mathbf{G}_2 \doteq \sum_{\Omega}\nabla I'^T\mathbf{R}\mathbf{R}^T\nabla I', \quad (5.13)$$

$$\mathbf{b}_1 \doteq \sum_{\Omega}\delta I\nabla I, \quad \mathbf{b}_2 \doteq \sum_{\Omega}\delta I'\nabla I'\mathbf{R} \quad (5.14)$$

Finally, we get:

$$\frac{1}{2}\left[\frac{\partial E}{\partial \mathbf{d}}\right] = (\mathbf{G}_1 + \mathbf{G}_2)\mathbf{d} - (\mathbf{b}_1 + \mathbf{b}_2) = 0 \quad (5.15)$$

$$\mathbf{d}_{opt} = (\mathbf{G}_1 + \mathbf{G}_2)^{-1}(\mathbf{b}_1 + \mathbf{b}_2) \quad (5.16)$$

The 2D Lucas-Kanade tracking method is usually performed in an iterative manner on a pyramid of images to tolerate for large displacements. The same can be done for the 3D Lucas-Kanade tracking. I used 4 levels of the pyramid. The "zeroth" level is the highest resolution (original) image. For each next level, the resolution is reduced twice in each dimension. Let $L = 1, 2...$ be a generic pyramid level. If point $\mathbf{m}$ is projected to a point (u_0, v_0) in the original image, then on level L it will be projected to $\frac{1}{2^L}(u_0, v_0)$. It means that the image gradient vector with respect to the 3D displacement $\mathbf{d}$ on level L will be different:

$$\nabla I_L = \frac{1}{2^L}\begin{bmatrix} I_{Lx} \\ I_{Ly} \\ I_{Lz} \end{bmatrix} = \frac{1}{2^L}\begin{bmatrix} \frac{\partial I_{t+1(L)}}{\partial u}\frac{f_1}{m_z} + 0 \\ 0 + \frac{\partial I_{t+1(L)}}{\partial v}\frac{f_1}{m_z} \\ -\frac{\partial I_{t+1(L)}}{\partial u}\frac{m_x f_1}{m_z^2} - \frac{\partial I_{t+1(L)}}{\partial v}\frac{m_y f_1}{m_z^2} \end{bmatrix} \quad (5.17)$$

The rest is the same for each level. We start from the most coarse level and on each level the displacement $\mathbf{d}$ is computed and then used as an initial guess on the next level.

5.3.2 Two head models and two ways of computing displacement

I tried two different head models: a cylinder and a flat rectangle. Also two different ways of computing a displacement were evaluated.

Cylinder head model, computing overall displacement for the whole head

In the first version of the 3D Lucas-Kanade tracker, the head is modelled as a cylinder. This tracker is described in Harguess *et al.* (2011) and Xiao *et al.* (2002). The template points are placed on the cylinder surface, so that around 100 points are visible at any moment in time. An example of the cylinder head model is shown in Figure 5.11. The equation 5.16 is solved for all the sample points at the same time and the overall head displacement is computed.

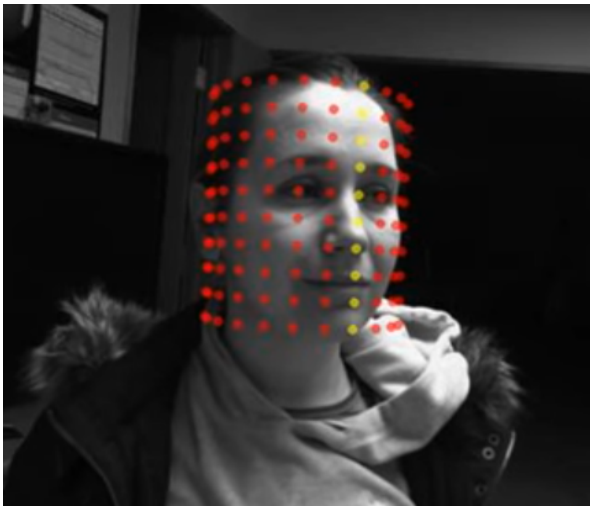

Figure 5.11: The set of points in the cylinder head model.

According to the surface geometry, the template points do not have a uniform density in the image. This affects their contribution. To take this into account, a set of weights is used. Suppose u is the projection of a head point X. θ is the angle between the surface normal at X and the direction from the head center to the camera center, as shown in Figure 5.12. We compute the pixel density weight by a quadratic function because a quadratic surface (cylinder) is used as the model:

$$w = \left(1 - min\left(|\theta_u|, \frac{\pi}{2}\right)\frac{2}{\pi}\right)^2 \tag{5.18}$$

When $\theta \geq \frac{\pi}{2}$, w is 0 because the point projection u is not visible. Smaller values of θ mean that u is closer to the template center and has lower density, so w is larger accordingly.

The 3D Lucas-Kanade tracking with the cylinder head model is usually used in videos, where the head occupies a large part of the camera viewing zone (head-and-

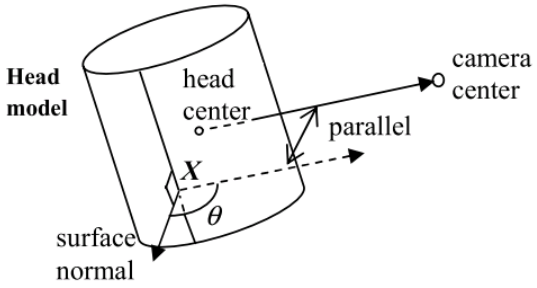

Figure 5.12: Angle θ between the surface normal and the direction from the head center to the camera center, which is used to determine the pixel density weight. The image is taken from Xiao *et al.* (2002) ©IEEE

shoulders videos). The head translation and rotation are usually estimated. I evaluated this method on the whole body view videos. In this type of video, the face is not large enough to estimate rotation accurately. Therefore, only head translation is estimated[1].

Flat face model, computing displacements for sample points separately

The 3D Lukas-Kanade tracking with the flat face model is one of my contributions. An example of the flat rectangle face model is shown in Figure 5.11. Another major difference of this method to the previous method is that a separate displacement is computed for a 4×4 pixel subwindow around each sample point. Then an average displacement is computed for the overall head displacement[2]. An example of the sample points' displacements is shown in Figure 5.14.

These two methods were compared experimentally. The experiment was performed on a short dataset of 5 videos with easy slow motions. Each video is 600 frames long. The results are shown in Figure 5.15. The second method with the flat rectangle face model and computing displacements separately for each point showed considerably better performance. Even though it looks like that the cylindrical model fits the head shape much better than the flat rectangle model, the model shape does not really play an important role because the tracked person is several meters away from the camera

[1] A short example of the implemented method can be seen in this video: https://youtu.be/QfJnwxOCsJo.

[2] A short example of the implemented method can be seen in this video: https://youtu.be/IarIt0d6zYs.

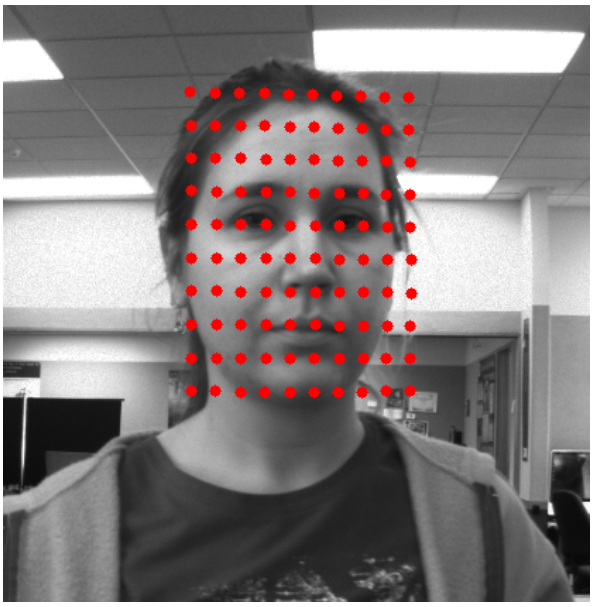

Figure 5.13: The set of points on the flat rectangle face model.

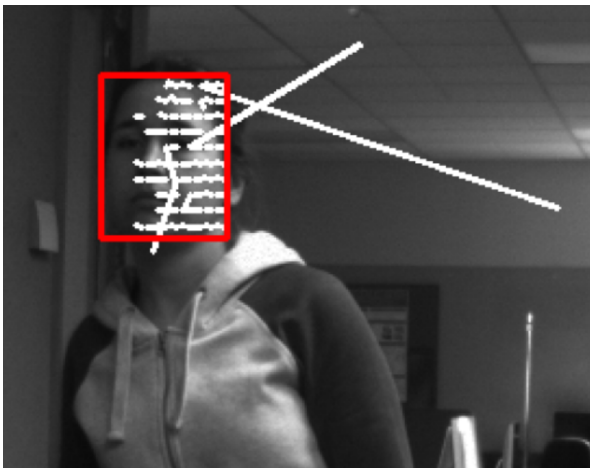

Figure 5.14: A separate displacement is computed for each sample point.

and the face is small.

5.3.3 Forward-backward tracking

As mentioned earlier, the main component of face adapted median flow is the median flow tracker. The median flow tracker is the Lucas-Kanade tracker with the forward-backward (FB) failure detection. This failure detection method was developed by the

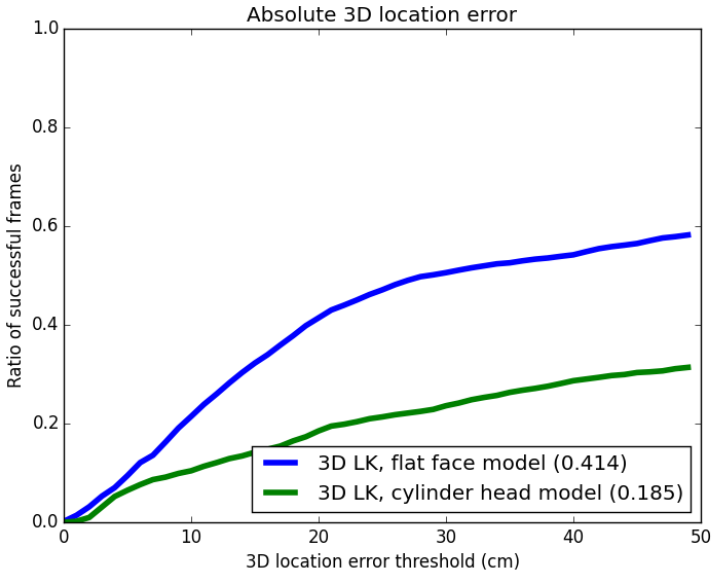

Figure 5.15: The location error plot for the 3D Lucas-Kanade tracker with flat model and the cylinder model.

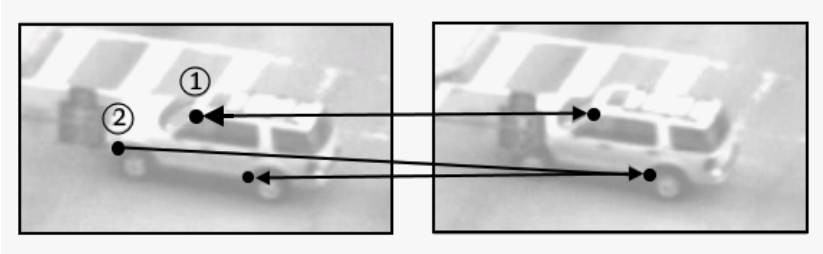

Figure 5.16: The FB error penalizes inconsistent trajectories. Point 1 is visible in both images, tracker works consistently forward and backward. Point 2 is occluded in the second image, forward and backward trajectories are inconsistent. The image is taken from Kalal *et al.* (2010b) ©IEEE.

author of TLD and described in Kalal *et al.* (2010b).

The FB method is based on the so called forward-backward consistency assumption that correct tracking should be independent of the direction of time-flow. Each point tracked from frame I_1 to frame I_2, is then tracked backwards from from frame I_2 to frame I_1. If it ends up in a location different to the initial one, this point is penalized. In Figure 5.16, the FB trajectory of point 1 is consistent, while point 2 ends up in a

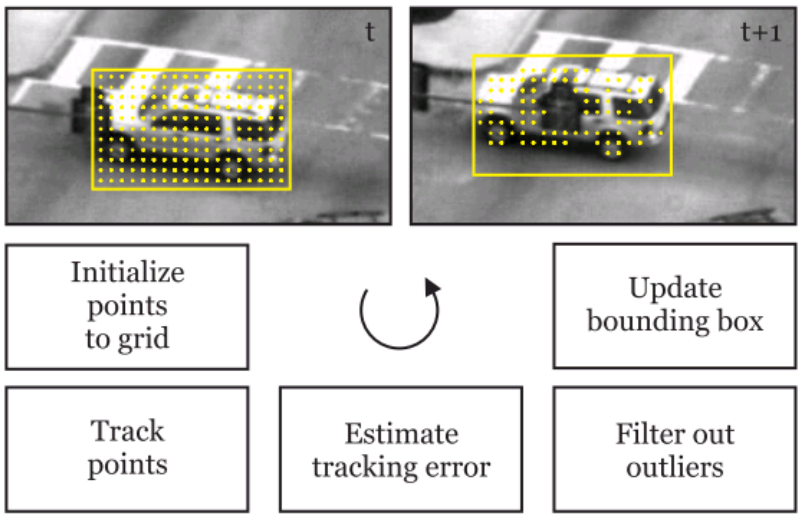

Figure 5.17: The Lucas-Kanade tracker accepts a bounding box and a pair of images. A number of points within the bounding box are tracked, their error is estimated and the outliers are filtered out. The remaining estimate the bounding box motion. The image is taken from Kalal *et al.* (2010b) ©IEEE.

different place to its initial location.

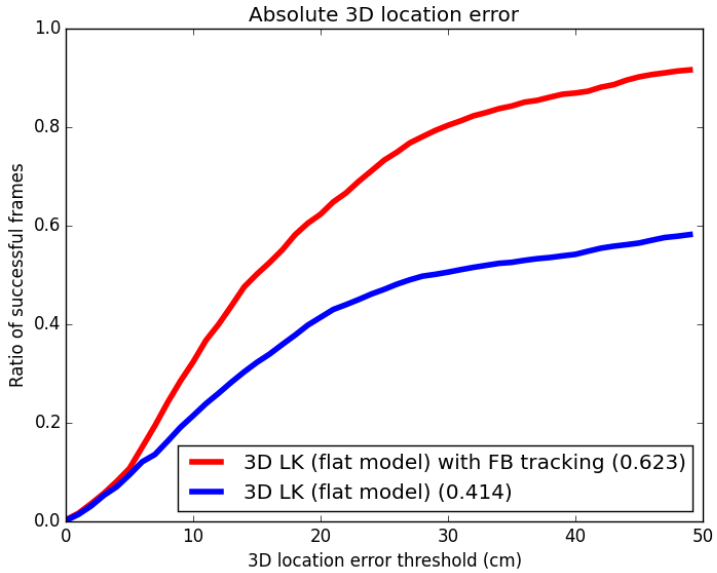

Figure 5.18: The location error plot for the 3D Lucas-Kanade tracker with FB tracking and without it.

This method can be used to modify the Lucas-Kanade tracker when displacement is computed for each sample point separately (as a sparse motion flow). The block-diagram of the Lucas-Kanade tracker with the FB error estimation is shown in Figure 5.17. The FB error can be computed as an Euclidean distance between the initial sample point location and the final sample point location after backward tracking. The best 50% of the sample points are retained, and used to estimate the target's new position.

The following experiment checked whether the FB error estimation improves the 3D Lucas-Kanade tracker with the flat rectangular face model. The performance of the trackers is shown in Figure 5.18. The FB error estimation improved results greatly. Following the author of the method, I will call the 3D Lucas-Kanade tracker with the FB tracking the 3D median flow tracker.

5.3.4 Comparing 2D and 3D median flow trackers

Finally, I explored whether the 3D median flow tracker is better than using two independent instances of the 2D median flow tracker and triangulating the result. Both trackers estimate displacements for all sample points separately and use the FB method to detect failures. The experiment was performed on the set of 23 pairs of high resolution stereo videos ranging from 600 to 3000 frames long. As our dataset is quite challenging and contains many rapid movements and full occlusions, the median flow trackers usually fail in the beginning. To overcome this problem, the tracking was reinitialized from the ground-truth data every 25 frames. This reinitialization frequency is equal to the optimal frequency of face redetection in the face adapted tracking methods (see Section 4.4.1).

The performance of the two methods is shown in Figure 5.19. The performance is very similar, but the 3D median flow tracker is slower. The frame rates of the two methods are shown in Figure 5.20.

It is possible that the 3D median flow tracker can outperform the 2D median flow tracker in the head-and-shoulders videos. When a face occupies a large part of the screen, there is more information to use. In the head-and-shoulders videos, the head rotation can be estimated and it can improve the overall tracking accuracy. However, for our task of the whole body view face tracking, the 2D median flow tracker does a better job. Its accuracy is similar to 3D median flow, but it is 1.5 times faster.

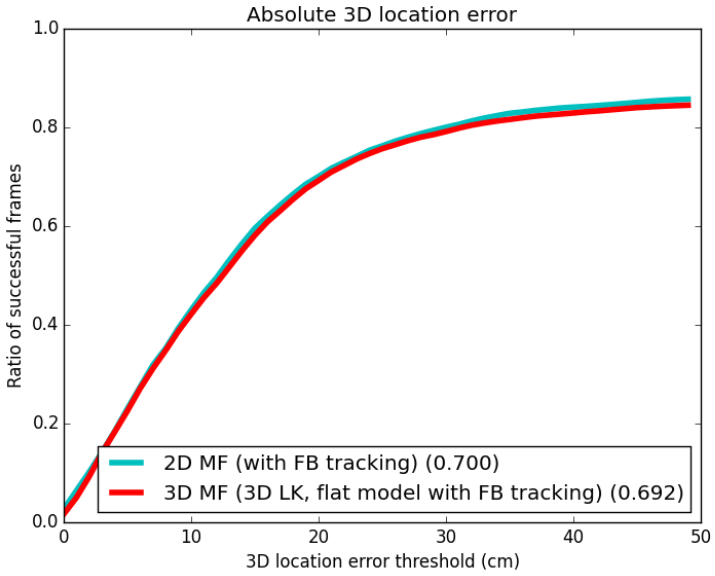

Figure 5.19: The location error plot for the 3D median flow and the 2D median flow.

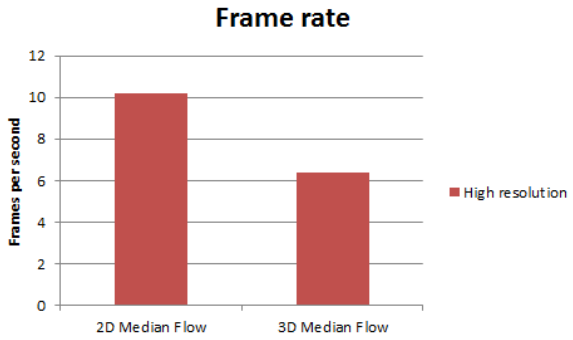

Figure 5.20: The frame rate plot for 3D median flow and 2D median flow.

5.4 Compliance with requirements

In this section, I evaluate how the resulting 3D face adapted median flow tracker complies with the requirements set for this research in Section 1.2.

- The resulting 3D face tracker works on feed from two luminance stereo cameras.
- It shows satisfactory accuracy in the range of 0.5 - 5 meters. The tracked person can move freely in this range, walk, run, jump and perform other actions.

- On low resolution videos (640×512) it can run at around 10 fps. On high resolution videos (1280×1024) the tracking accuracy is higher, but the frame rate is around 4 fps. That is not quite enough for tracking in real time applications, but the speed could be improved by the code optimization and using GPU.

- The resulting 3D face adapted median flow gives a reliable long-term tracking of faces. It can recover after full-occlusions and other tracking failures.

- It was evaluated on an extensive dataset in an indoor environment with a cluttered background and multiple occlusions and changes in illumination (for the list of all challenges in the evaluation dataset, see Table 3.1). It shows reasonable accuracy on this dataset.

5.5 Conclusion

This chapter was concerned with converting 2D face adapted median flow into 3D tracking in the most efficient way. The straightforward way is to run two independent instances of the 2D tracker and then triangulate the result. I looked into ways of using stereo information to facilitate 3D tracking. A novel method of checking the consistency of the target 3D position and size was developed and evaluated. Also I examined sharing the collections of positive and negative examples between the 2D trackers. However, this does not improve the tracking performance. Several 3D Lucas-Kanade trackers were described and evaluated. I developed and evaluated my own version of the 3D Lukas-Kanade tracker with the flat face model. For the whole body view 3D face tracking, running two independent instances of the 2D median flow trackers and triangulating the results gives a higher frame rate and a slightly better accuracy.

In general, the resulting 3D face adapted median flow gives a reliable long-term tracking of faces and complies with the requiremnts set in Section 1.2. It can recover after full-occlusions and other tracking failures. On low resolution videos (640×512) it can run at around 10 fps and with reasonable accuracy. On high resolution videos (1280×1024) the tracking accuracy is higher, but the frame rate is around 4 fps. That is not quite enough for tracking in real time, but the speed could be improved by the code optimization and using GPU.

Chapter 6

3D Face tracking in fisheye video

Note: Some portions of this chapter were published in Mikhisor *et al.* (2014).

Stereo cameras with wide lenses give a wider joint field of view which can be useful in many practical applications. For all the experiments described in previous chapters, I used image sequences created by cameras with normal perspective lenses with the $79^\circ \times 59^\circ$ field of view (horizontal dimension $\times$ vertical dimension). A much wider field of view can be achieved by using fisheye lenses with the $180^\circ \times 180^\circ$ field of view. A comparison of the joint viewing zones and resulting images for normal perspective cameras and fisheye cameras is illustrated in Figure 6.1. The downside of a fisheye camera is that all objects are smaller in the resulting images and the distortion gets significant close to the edges. Because a camera with a fisheye lens covers a substantially larger space compared to a camera with a normal lens, each pixel in a resulting fisheye image covers a larger volume in space and everything looks smaller in this image. The closer a pixel is to a fisheye image edge, the larger the volume in space that is represented by this pixel and the more distorted and squished are objects that are shown in this part of the fisheye image.

Using fisheye cameras allows an extra large viewing zone, but higher distortion and smaller objects' sizes pose additional challenges for tracking. In this chapter I will evaluate several 3D face tracking methods on fisheye stereo dataset. I will compare 3D face adapted median flow and 3D face adapted TLD trackers developed in Chapter 5 with a simple colour particle filter and a 3D version of this particle filter. Also these methods will be compared with the Kinect face tracking application. The contributions of this chapter are the following. I demonstrate how a visual tracking method can be compared to a Kinect based tracking method. Also I show that a 3D particle filter

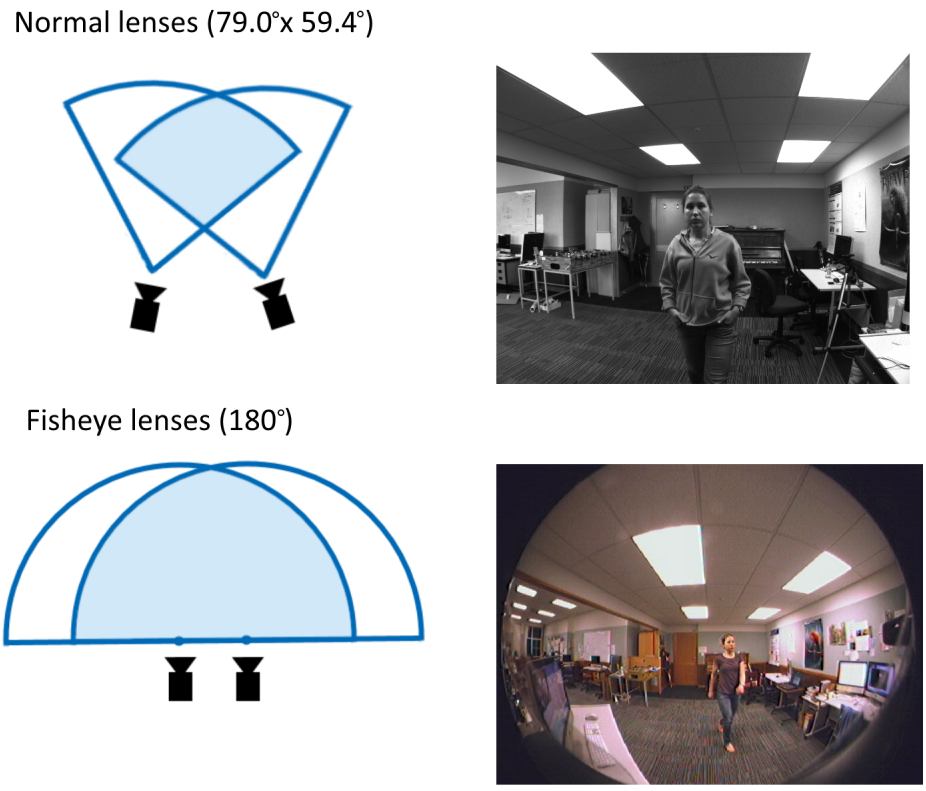

Figure 6.1: Top row: the joint viewing zone and an image example for the cameras with perspective lenses. Bottom row: the joint viewing zone and an image example for the cameras with fisheye lenses.

outperforms a 2D particle filter for the task of 3D tracking.

Colour particle filters were not evaluated in Chapter 5 because they rely heavily on the color information and do not work properly in my main grey-scale dataset used in the experiments in Chapter 5. Kinect was not compared to the other 3D tracking methods in Chapter 5 because prerecording the Kinect data is a tedious and time consuming process and it was not done for my main stereo dataset.

Section 6.1 describes how to record a dataset to compare a Kinect based tracking method to visual tracking methods that work on ordinary luminance video. In Section 6.2 2D and 3D particle filters are introduced. In Section 6.3 five different tracking methods are evaluated: a 3D particle filter, a 2D particle filter with triangulation, 3D face adapted median flow, 3D face adapted TLD (these four methods run on the fisheye stereo video) and the Kinect face tracking application.

6.1 Comparing Kinect to visual tracking methods

Kinect is the most popular and accessible depth sensor (Han *et al.*, 2013). It is used in many 3D face tracking applications (see Section 2.1.1). One of the goals of this chapter is to compare the performance of the Kinect 3D face tracking application[1] and other visual 3D face tracking methods. Comparing a Kinect based method to other tracking methods running on normal luminance cameras is not a straight forward task. The standard way of comparing trackers is to obtain a dataset (record, or download from the internet) and then run the evaluated methods as many times as you need. However, to compare visual tracking methods to Kinect, the Kinect face tracking results have to be recorded at the same time as recording the dataset using luminance cameras. For the experiment described in this chapter, I recorded only 3D positions of the face detected by Kinect. However, it is possible to record the face size and orientation as well if needed. To be able to compare the tracking results, Kinect and the stereo cameras have to be synchronized and stereo calibrated. These two procedures are explained in this section.

Visual cameras and Kinect calibration

Before recording stereo videos and the face tracking data from Kinect, Kinect and the cameras have to be calibrated. Kinect returns the head position and orientation in the coordinate system of its depth sensor. Visual tracking methods return the head position in the coordinate system of one of the two stereo cameras. To compare these results, stereo calibration for the depth sensor and one of the fisheye cameras has to be performed. In other words, we need to find the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ required to transform the fisheye camera coordinate system into the coordinate system of the Kinect depth sensor. These two parameters can be found in the form of the essential matrix $\mathbf{E} = \mathbf{R}[\mathbf{t}]_{\mathbf{x}}$, where $[\mathbf{t}]_{\mathbf{x}}$ is the matrix representation of the cross product with $\mathbf{t}$. To estimate the essential matrix, I used the normalized eight-point algorithm (Hartley, 1997). The corresponding points in the depth sensor images and the fisheye camera images were selected manually (see an example in Figure 6.2). After selecting 120 corresponding points and computing the calibration parameters, I got the reprojection error of 2.26 pixels which is acceptable. It is hard to achieve higher accuracy because both the fisheye image and the Kinect depth image are noisy and

[1] The Microsoft Face Tracking Software Development Kit for Kinect for Windows https://msdn.microsoft.com/en-us/library/jj130970.aspx

have low resolution. The error of 2.26 pixels is equivalent to the 5 cm error at the distance of 3 meters.

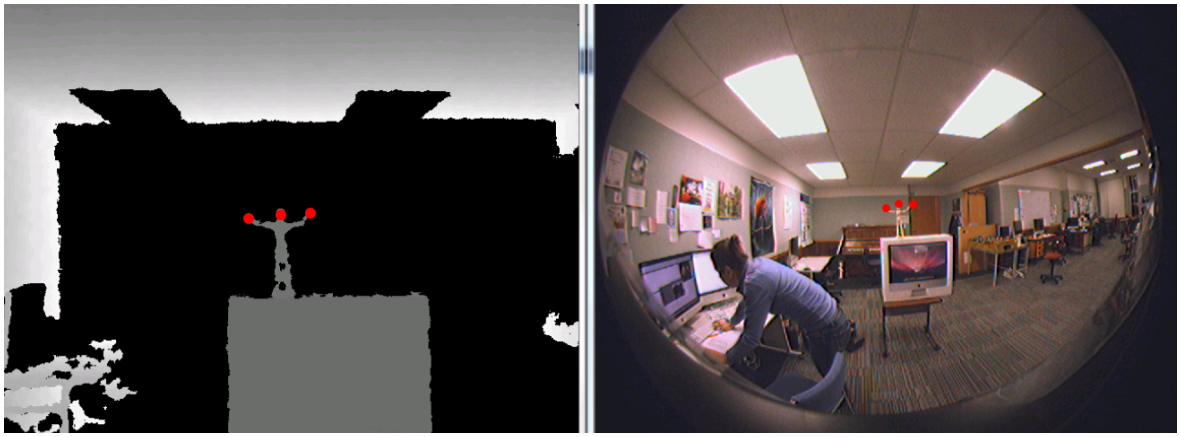

Figure 6.2: Selecting corresponding points in the depth sensor image and the fisheye camera image. The three corresponding points (the head and hands of the wooden figure) are marked by red circles in both views.

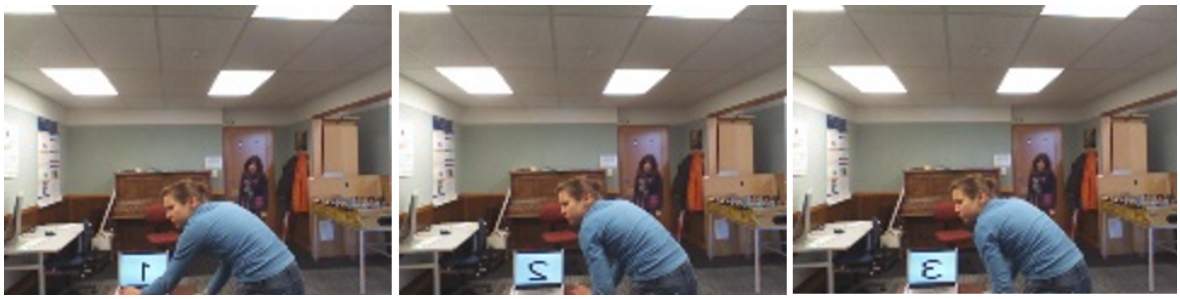

Figure 6.3: A synchronization sequence was recorded in the start and end of each evaluation video to help synchronizing the Kinect data and the fisheye video.

Visual cameras and Kinect synchronization

Another challenge when comparing a Kinect based tracking method to other visual tracking methods is two synchronize the visual dataset and the Kinect tracking data. This means that the recording of both stereo videos and the Kinect data should start at the same time. The Unibrain cameras that I used for recoding fisheye videos have a slightly uneven frame rate because some frames take longer to be recorded to the

hard drive. Even though nominally Kinect and the cameras work at the same frame rate of 30 frames per second, in the end the tracking data recorded from Kinect does not match the fisheye video. To overcome this problem, the end of recoding the stereo datasets and the Kinect tracking data should be synchronized as well. Kinect and the cameras cannot be synchronized on the software level because for performance reasons these devices run on separate computers. Instead, I synchronised the recording manually. In the beginning and in the end of each evaluation video, I recorded a short synchronization sequence (some changing numbers shown on a laptop, see Figure 6.3 for an illustration) on the Unibrain fisheye cameras and on the Kinect color stream. After finishing the recording, I had to synchronize the Kinect color stream and the fisheye cameras recordings and adjust the ground truth data recorded from Kinect. To do that, I looked though the cameras' streams and the Kinect color stream and found the frames that show exactly the same image in the synchronization sequence shown on a laptop (for example, the moment when number "3" appears). This is done for the start and for the end of a recording. All the cameras' frames and the Kinect tracking data before the start and after the end synchronization point were discarded. After that I had to make sure that both stereo cameras have exactly the same number of frames between the start and the end syncronization points and this number matches the number of entries in the Kinect tracking data. The results were checked visually by projecting the Kinect ground truth data into the fisheye video[1]. This is an approximate method of synchronization, but it gives pretty good results.

After the calibration and synchronization of the Kinect tracking data and the visual dataset, any visual tracking method can be evaluated on the visual dataset and compared to Kinect. The results of this experiment are described in Section 6.3. Stereo calibrating and synchronizing Kinect with luminance cameras is a tedious and time consuming process. For this reason, I did not record the Kinect data for my main dataset with normal cameras and Kinect was not evaluated on it and not compared to the other methods in Chapter 5.

6.2 Using a particle filter for 3D tracking

Besides Kinect 3D face tracking, 3D face adapted median flow and 3D face adapted TLD, I evaluated the conventional particle filter with color histograms for the task of 3D

[1] An example of Kinect data projected to the fisheye video can be found at https://youtu.be/nUQfXrb35iM.

tracking in stereo fisheye video. I decided to try out a simple and popular approach of 2D particle filtering and see if extending it into 3D can give better performance results. Colour particle filters were not evaluated in Chapter 5 because they rely heavily on the color information and do not work properly in my main grey-scale dataset used in the experiments in Chapter 5.

Particle filtering, also known as the Condensation algorithm (Isard and Blake, 1998) and bootstrap filter (Gordon *et al.*, 1993), has been proven to be a powerful and reliable tool for nonlinear systems (Carpenter *et al.* 1999). Particle filtering is an established technique because of its inherent property to allow fusion of different sensor data, to account for different uncertainties, to cope with data association problems when multiple targets are tracked with multiple sensors and to incorporate constraints.

The particle filter framework has been used in many people tracking algorithms (e.g Ali and Dailey 2012, Maggio 2007, Chang *et al.* 2005), including multi-camera tracking (e.g. Nummiaro *et al.* 2003, Wang *et al.* 2005, Li *et al.* 2012). There are several ways of using particle filters in a multi-camera environment: independent 2D particle filters, collaborating 2D particle filters, a 3D particle filter or a combination of 2D and 3D particle filters.

Nummiaro *et al.* (2003) describe collaborating 2D particle filters. When the tracked object is lost in one camera view, the epipolar constraint is used to estimate the target position from the other camera view. Wang *et al.* (2005) estimate the target position using the view that provides the most likely observations. In a sense this method switches observation models from one model to another.

In other papers a homography matrix is computed to align the ground plane of different cameras' views. Using this homography, it is possible to project some hypotheses from the 2D particle filter of one camera to the 2D particle filter of the other camera and make the two trackers collaborate (Li *et al.*, 2012). Du and Piater (2007) and Sunderrajan and Manjunath (2013) use a separate 2D particle filter in the ground plane together with 2D particle filters in each camera view. The main complication in using the homography of the ground plane is that usually the position of the person's feet is used as the particle position and it is often difficult to determine accurately. In the case of head tracking the ground plane homography is not useful at all because the head is too far from the feet.

One way to avoid using the ground plane homography is to use a 3D particle filter (Kobayashi *et al.* 2006,Nickel and Gehrig 2005). In a 3D particle filter, the tracked target state is represented as a 3D position in space. Each particle is projected into

each camera's image plane where the observations are made.

The particle filter framework

Using particle filters for visual tracking is described in the paper of Isard and Blake (1998). The basic particle filter framework is described further. The state of a tracked object at time t is described by the vector x_t, while the vector Z_t contains all the observations $\{z_1 \dots z_t\}$ up to time t. To estimate the current state x_t of the tracked object, we build the posterior probability distribution $p(x_t|Z_t)$.

The probability distribution is represented by a set of N samples $\{x_t^i\}_{i=1\dots N}$ (also called particles) with associated importance weights π_t^i. A particle weight is proportional to the likelihood that this particle coincides with the tracked object at that moment.

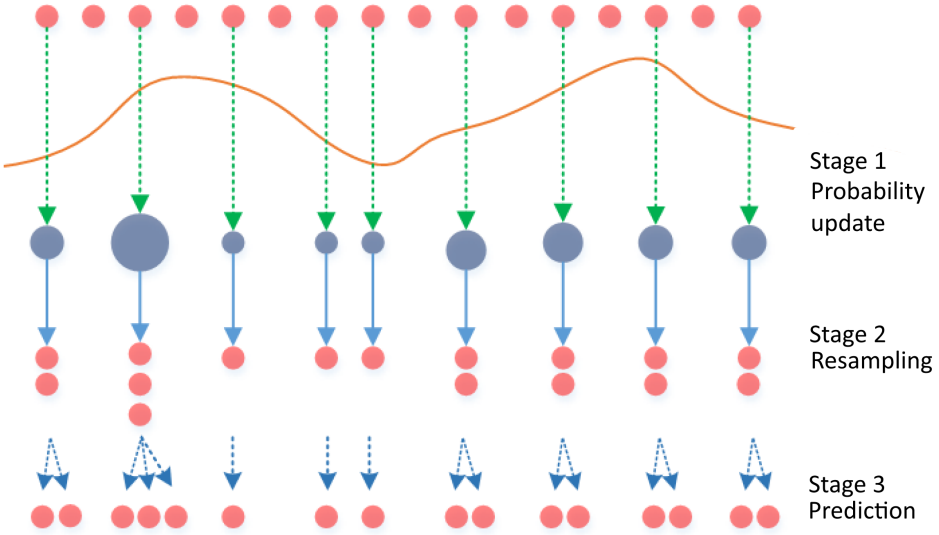

Figure 6.4: Three stages of the particle filter framework. The image is taken from Li *et al.* (2016).

The particle filter framework consists of essentially three stages: probability update, resampling according to the probability, and prediction according to the dynamical model (see Figure 6.4). When a new frame arrives, we make a new measurement z_t and update the probability distribution using Bayes rule (Stage 1):

$$p(x_t|z_t) = \frac{p(z_t|x_t)p(x_t|Z_{t-1})}{p(z_t|Z_{t-1})}. \tag{6.1}$$

Then the particles are resampled according to their probability (Stage 2). After that during the prediction stage (Stage 3), we use a probabilistic dynamical model $p(x_t|x_{t-1})$ to predict the expected motion of particles between time steps:

$$p(x_t|Z_{t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|Z_{t-1})dx_{t-1}. \tag{6.2}$$

Colour histograms provide an efficient feature for estimating a particle likelihood because they are robust to rotation, partial occlusion and scale invariant and computationally efficient (Pérez *et al.*, 2002). During the initialization step, the target histogram q of the face region is computed. Then for each particle, the histogram over its region p_{x_t} is computed. To estimate the particle likelihood, the Bhattacharyya coefficient is used:

$$\rho[p_{x_t}, q] = \sum_{u=1}^{m} \sqrt{p_{x_t}^{(u)} q^{(u)}}, \tag{6.3}$$

where m is the number of bins for the histograms, and $p_{x_t}^{(u)}$ is a value contained in the u^{th} bin of the histogram p_{x_t}.

2D particle filter for 3D head tracking

In the case of a 2D particle filter for head tracking[1], each particle is an ellipsoid and is characterized by its 2D position in the image and half-axes. For 3D head tracking using a 2D particle filter, two instances of the tracker run independently in each video stream. Then the probability is estimated and the particle with the highest probability is found in each view. The positions of the two particles with the highest probabilities are then triangulated to find the 3D head position.

3D particle filter for 3D head tracking

When running two independent instances of the 2D particle filter, the stereo dependencies are not used. To take advantage of the stereo dependencies, I developed the 3D particle filter. This algorithm is one of the contributions of this thesis. In the case of the 3D particle filter, each particle is a 3D position in space. All the stages apart from the observation stage are the same as for the 2D particle filter. At the probability update stage also known as the observation stage, each particle is projected to the image planes of both cameras as shown in Figure 6.5. I assume that the head height is

[1] A video example showing 2D particle filter in fisheye video can be found at http://youtu.be/XoEQdjCdZns.

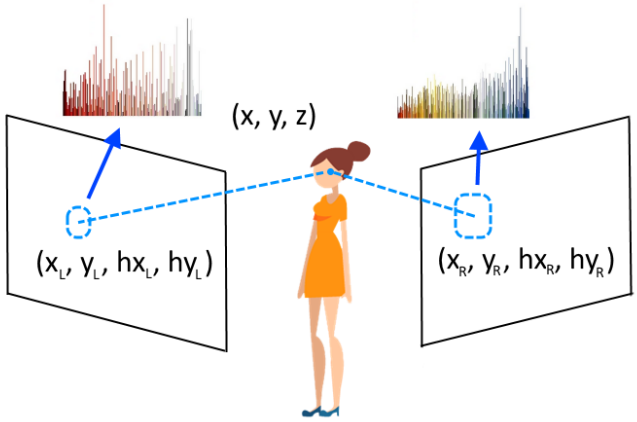

Figure 6.5: For each 3D particle, a 3D position is projected to the image plane and then an ellipsoid of the size that corresponds to 0.2 meters in 3D world, is created.

Figure 6.6: A visualization of 100 particles, projected to one of the cameras view. Ellipses with lighter colour have higher likelihood. Black ellipses have very small likelihood.

0.2 meters (from top of head to bottom of chin). By projecting the particle 3D position to the image plane and computing the corresponding head height in pixels, I get a face ellipsoid as in the case of the 2D particle filter. Figure 6.6 shows an example of 100 particles projected to one view. Then for each particle, I compute its histogram and compare it with the target histogram. As a result of the probability update step, I get two likelihood weights: one from each camera. The final weight for each particle

is the product of these two weights. After computing the weights, the particle set is resampled, and then the distribution is updated according to the 3D motion model[1].

6.3 Comparing 3D trackers on fisheye video and Kinect face tracking

In the experiment described in this section, I compared 3D face adapted TLD, 3D face adapted median flow, the 3D particle filter and a pair of 2D particle filters with triangulation on fisheye stereo videos and the Kinect 3D face tracking application. I recorded 10 pairs of videos of different complexity using two low resolution cameras with fisheye lenses (the Omnitech Robotics ORIFL190-3 fisheye lens with a field of view of 190°). All videos are about 2000-3000 frames long. The videos feature different people performing different actions (walking, running, jumping, rotating, exercising, dancing, etc.) and moving freely in the range of 0.5 - 5.0 meters from the camera. Some videos show 2 or 3 people at the same time.

This experiment was intended to test some scenarios of using an interactive display. However, it is more a qualitative evaluation rather than a proper user study of 3D face tracking for an interactive display. A proper user study is not in the scope of this research. One of the goals of this experiment is to show that visual tracking methods on fisheye stereo video can track a person in a larger viewing zone than Kinect. The Kinect viewing zone is smaller than the joint viewing zone of the two fisheye cameras. Both Kinect V1 and V2 have a very short depth range of 3.5 m and limited viewing angles (see Table 6.3). Kinect V1 was used in this experiment. Approximately 50% of all frames in the fisheye datasets, the tracked person is out of the Kinect viewing zone.

Factors	**Kinect V1**	**Kinect V2**
Hor. field of view	57°	70°
Ver. field of view	43°	60°
Depth sensor	1.8 - 3.5 m	1.3 - 3.5 m

Table 6.1: Kinect field of view.

The results of the experiment are shown in Figure 6.7. The 3D particle filter outperforms the paired 2D particle filters. This shows that using stereo dependencies

[1] A video example showing 3D particle filter in fisheye stereo video can be found at http://youtu.be/ec289yuCuwQ.

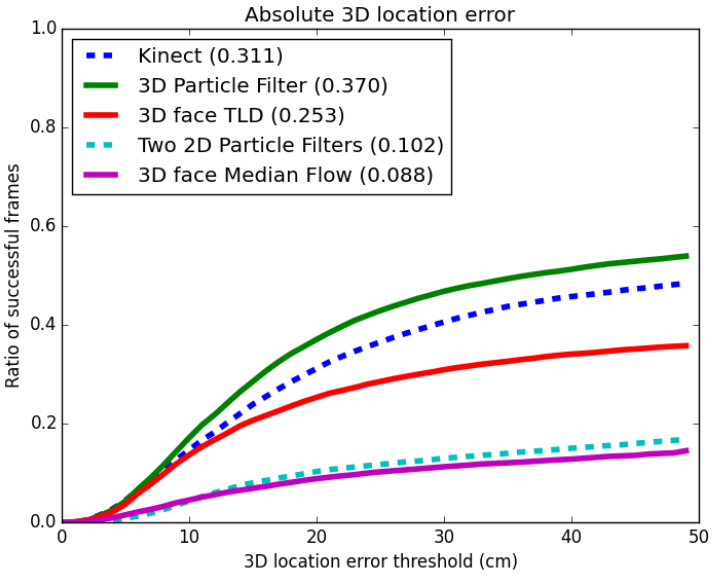

Figure 6.7: The location error plot for the five 3D trackers evaluated on the fisheye stereo dataset in the unconstrained viewing zone.

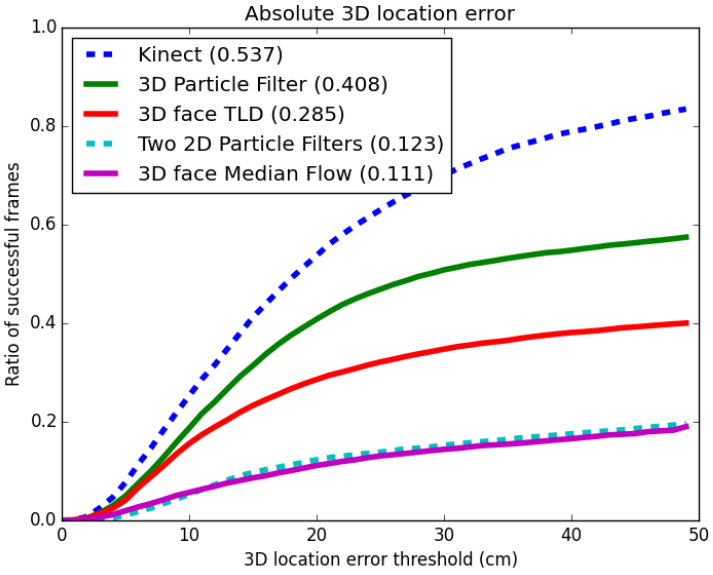

Figure 6.8: The location error plot for the five 3D trackers evaluated on the fisheye stereo dataset in the Kinect viewing zone.

can improve 3D tracking dramatically. The 3D particle filter also shows superior results over Kinect, 3D face adapted TLD and face adapted median flow. The Kinect face

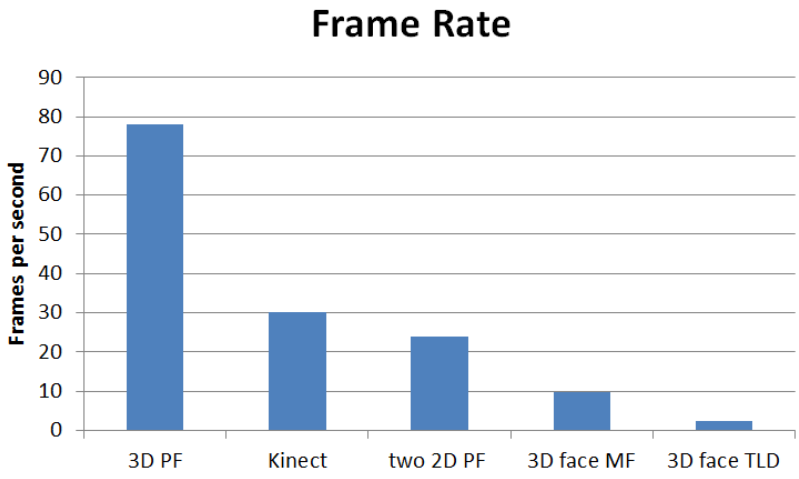

Figure 6.9: The frame rate plot for the five 3D trackers evaluated on the fisheye stereo dataset.

tracking performance is not high because approximately half of the time the target is out of the Kinect viewing zone. In Figure 6.8 the results of this experiment are shown only for those frames where the target is visible by Kinect. In this plot the Kinect face tracking shows superior results other all the other trackers.

Several questions arise from the results of this experiment. 2D and 3D versions of face adapted median flow showed superior results in videos made with normal cameras with resolution equal or higher than 640×480 (for example see Figure 4.37 and Figure 5.2). In the fisheye video the 3D face adapted median flow performance is dramatically low and is close to the performance of the paired 2D particle filters (Figure 6.7). The 3D face adapted TLD performance is also lower than on normal medium or high resolution videos. These two trackers are based on the Viola Jones detector. The Viola Jones detector recall is low for small faces. The TLD detector accuracy is also lower for small faces. In the stereo fisheye dataset the average head size is very small (around 10-15 pixels). That is why the performance of these two trackers is so low in this experiment compared to the experiments described in the previous chapters. In future work, it would be interesting to find experimentally what are the face size limits for using face adapted TLD and face adapted median flow and if there are any other limitations on using these trackers.

Also it is interesting to note that using the stereo dependencies substantially improved the 3D particle filter performance compared to the paired 2D paired particle filters. In Chapter 5, I tried to find a way how to use stereo dependences to improve

the 3D face adapted median flow. I got a minor improvement by using the epipolar constraint (see Figure 5.6), but this is not as great as the improvement shown by the 3D particle filter over the paired 2D particle filters. It is a challenge for future work to find the way to use stereo information to get a similar improvement in 3D face adapted median flow. Also it would be interesting to compare 3D particle filter and 3D face adapted median flow on normal high resolution video. I cannot do that on my high resolution dataset, because it is grey scale and the color particle filter does not work properly on it.

The frame rates of the tested trackers are shown in Figure 6.9. The 3D particle filter is the fastest and is almost three times faster than the paired instances of the 2D particle filter. Even though the 3D particle filter runs in real time and shows the best performance, its accuracy is pretty low. In only 40% of the frames are the tracked head positions more than 20 cm away from the real head position. However, I used the simplest version of the particle filter frame with color histograms. There are many more complex versions of 2D particles filters (for example a cascade particle filter with discriminative observers by Li *et al.* 2008) that can improve the 3D particle filter and make its accuracy on fisheye stereo video acceptable.

6.4 Conclusion

In this chapter, I demonstrated how a Kinect based tracking method can be compared to visual tracking methods running on luminance stereo video. Five different trackers were compared: the 3D particle filter, two 2D particle filters with triangulation, 3D face adapted median flow, 3D face adapted TLD and the Kinect face tracking application. The four visual tracking methods were evaluated on low resolution fisheye stereo videos. The 3D particle filter shows the best results out of all five trackers.

Chapter 7

Conclusion

Even though there is an emerging interest in 3D tracking and many new applications based on it appear regularly, this field is not so well studied and described in the literature as 2D tracking. A large variety of different learning methods, appearance models and tracking features that have been thoroughly evaluated in many different combinations in 2D tracking papers (Yang *et al.* 2011, Li *et al.* 2013), have not yet been tried out in 3D tracking. Also for 2D tracking recently, it has become a widely used practice to make the source code available to allow easy evaluation and comparison. There are several projects (e.g. http://www.votchallenge.net and http://cvlab.hanyang.ac.kr/tracker_benchmark/) that maintain state of the art 2D trackers source code repositories as well as easy to use and interpret evaluation metrics and categorized annotated evaluation datasets. This allows us to choose a 2D tracking algorithm that is most suitable for a specific application or compare a new algorithm to state of the art 2D trackers. For 3D tracking there are hardly any trackers or evaluation datasets available online. For this reason, I followed a non-classical approach of developing a 3D tracking algorithm: instead of computing a disparity map or using stereo feature matching techniques, I developed a 2D face tracking algorithm in one view and then extended it into stereo tracking.

Even though there are many successful generic 2D trackers available online and these trackers are designed to perform reasonably well on all sorts of different scenarios, all trackers have their strengths and weaknesses. Therefore, instead of using results of evaluation experiments conducted on public datasets (Kristan *et al.* 2015, Wu *et al.* 2015), I recorded and annotated my own extensive dataset specifically for 2D face tracking in whole body view video. By evaluating 17 state of the art 2D trackers on my dataset, I found out that TLD suits best of all for long term 2D face tracking in whole

body view video. The reason for that is that it handles robustly the two challenges that are the most prominent in my dataset: drastic scale change and frequent prolonged occlusions.

Furthermore, I showed that giving up tracking generality can improve accuracy greatly. This is often useful because for a large number of tracking applications, the visual class of the object of interest is known. In this research, I focus on tracking faces. I designed and implemented the adaptation of generic trackers for tracking exclusively faces. Generic trackers adapted for face tracking using this method showed considerable accuracy improvement. I developed close to real time face adaptation of the median flow tracker (Kalal *et al.*, 2010b). Originally, the median flow tracker is a part of TLD. However, my experiments showed that face adapted median flow outperforms face adapted TLD. Also, I developed the sliding window detection method that allows to run face adapted median flow close to real time on low resolution videos.

After developing a robust 2D face tracker, I explored different ways of extending 2D tracking into 3D and developed a method of using the epipolar constraint to check consistency of 3D tracking results. This method allows to detect tracking failures early and improves overall 3D tracking accuracy. The final 3D face adapted median flow tracker shows good tracking results and reasonable speed that could be improved by code parallelisation and optimization.

Using cameras with fisheye lenses gives a larger field of view and a smoother experience when performing 3D face tracking for real time interactive applications. However, cameras with wide angle lenses also have high distortion and information density issues that pose additional challenges for tracking. Five different trackers were compared: the 3D particle filter, two 2D particle filters with triangulation, 3D face adapted median flow, 3D face adapted TLD and the Kinect face tracking application. The four visual tracking methods were evaluated on low resolution fisheye stereo videos. I demonstrated how a Kinect based tracking method can be compared to visual tracking methods running on luminance stereo video. The 3D particle filter shows the best results out of all five trackers.

7.1 Future work

The contributions of this thesis raise the following issues for future research:

- One important assumption that was held throughout the work presented here is that state of the art 2D tracking methods can improve 3D tracking performance.

Therefore, instead of using standard stereo matching techniques, I explored a 2D tracking algorithm and different ways to extend it into 3D space. The only stereo technique used in the final 3D face adapted median flow is the epipolar constraint to check the 3D tracking result consistency. However, the addition of such popular 3D tracking techniques as computing depth map for foreground segmentation, creating an online updated background model, plan view statistics, and special points (edges and corners) stereo matching are likely to improve tracking results.

- As each face is attached to a body, information contained in the body appearance can be used to improve tracking performance. Local key-points under the tracked face with consistent co-occurrence and motion correlation could be extracted and used to help the tracker differentiate the target from a background and from other people faces. These key-points can be treated as 'supporters', as described in the work of Dinh *et al.* (2011).

- Detecting faces with different in plane and out of plane rotations can help to further improve the resulting 3D tracking accuracy. The rotation invariant multi-view face detector developed by Bo Wu *et al.* (2004) could be used for this purpose.

- For interactive applications such as head coupled display, smooth tracking is often more important than precise 3D position. Currently, the 3D face adapted median flow can be jerky sometimes, when the face detector corrects the tracker. Some trajectory smoothing could be introduced to address this problem.

- Finally, it would be interesting to perform a user study of the tracking algorithm in some application such as a personal 3D display or a public display and see if the 3D tracking algorithm is accurate and robust enough to provide an adequate experience to users.

References

Abate, A. F., Nappi, M., Riccio, D., and Sabatino, G. (2007). 2D and 3D face recognition: A survey. *Pattern Recognition Letters*, *28*(14), 1885–1906.

Abbaspour, M. J., Yazdi, M., and Shirazi, M.-a. M. (2014). Robust approach for people detection and tracking by stereo vision. In *7th International Symposium on Telecommunications (IST).*, 326–331.

Ali, I. and Dailey, M. N. (2012). Multiple human tracking in high-density crowds. *Image and Vision Computing*, *30*(12), 966–977.

Argyros, A. A. and Lourakis, M. I. (2004). Three-dimensional tracking of multiple skin-colored regions by a moving stereoscopic system. *Applied optics*, *43*(2), 366–378.

Asthana, A., Zafeiriou, S., Cheng, S., and Pantic, M. (2014). Incremental Face Alignment in the Wild. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1859–1866.

Babenko, B., Yang, M.-H., and Belongie, S. (2011). Robust Object Tracking with Online Multiple Instance Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(8), 1619–1632.

Bagdanov, A. D., Del Bimbo, A., and Nunziati, W. (2006). Improving evidential quality of surveillance imagery through active face tracking. *Proceedings - International Conference on Pattern Recognition*, *3*(September), 1200–1203.

Bahadori, S., Iocchi, L., Leone, G. R., Nardi, D., and Scozzafava, L. (2007). Real-time people localization and tracking through fixed stereo vision. *Applied Intelligence*, *26*(2), 83–97.

Bianco, S., Ciocca, G., Napoletano, P., and Schettini, R. (2015). An interactive tool for manual, semi-automatic and automatic video annotation. *Computer Vision and Image Understanding*, *131*, 88–99.

Bo Wu, Haizhou Ai, Chang Huang, and Shihong Lao (2004). Fast rotation invariant multi-view face detection based on real adaboost. *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.*, 79–84.

Boschini, M., Poggi, M., and Mattoccia, S. (2016). Improving the reliability of 3D people tracking system by means of deep-learning. In *2016 International Conference on 3D Imaging (IC3D)*, 1–8. IEEE.

Bouguet, J.-Y. (2001). Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *ntel Corporation*, *5*(4), 1–10.

Brar, R. S., Surman, P., Sexton, I., and Hopf, K. (2010). Multi-user glasses free 3D display using an optical array. *3DTV-CON 2010: The True Vision - Capture, Transmission and Display of 3D Video*, *2*(1), 4–7.

Butt, M. U. and Morris, J. (2011). Precise Tracking using High Resolution Real-time Stereo. In *Proc. 26th Image and Vision Computing New Zealand Conf.(IVCNZ 2011)*, 143–148.

Cai, L., He, L., Xu, Y., Zhao, Y., and Yang, X. (2010). Multi-object detection and tracking by stereo vision. *Pattern Recognition*, *43*(12), 4028–4041.

Cannons, K. (2008). A Review of Visual Tracking. *Dept. Comput. Sci. Eng., York Univ., Toronto, Canada, Tech. Rep. CSE-2008-07*, 242.

Carpenter, J., Clifford, P., and Fearnhead, P. (1999). Improved particle filter for nonlinear problems. *IEE Proceedings - Radar, Sonar and Navigation*, *146*(1), 2.

Castrillón, M., Déniz, O., Hernández, D., and Lorenzo, J. (2010). A comparison of face and facial feature detectors based on the Viola-Jones general object detection framework. *Machine Vision and Applications*, *22*(3), 481–494.

Čehovin, L., Leonardis, A., and Kristan, M. (2015). Visual object tracking performance measures revisited. *IEEE Transactions on Image Processing*, *25*(3), 1–14.

Chang, C. and Chatterjee, S. (1992). Quantization error analysis in stereo vision. *[1992] Conference Record of the Twenty-Sixth Asilomar Conference on Signals, Systems & Computers*, 1037–1041.

Chang, C., Member, S., and Ansari, R. (2005). Kernel Particle Filter for Visual Tracking. *IEEE signal processing letters*, *12*(3), 242–245.

Chang, F. and Chen, C. J. (2003). A component-labeling algorithm using contour tracing technique. *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 741–745.

Checka, N., Wilson, K., Rangarajan, V., and Darrell, T. (2003). A Probabilistic Framework for Multi-modal Multi-Person Tracking. In *2003 Conference on Computer Vision and Pattern Recognition Workshop*, 100–100. IEEE.

Cheng, Y. (1995). Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *17*(8), 790–799.

Choi, C., Ahn, J., Lee, S., and Byun, H. (2006). Disparity Weighted Histogram-Based Object Tracking for Mobile Robot Systems. *Advances in Artificial Reality and Tele-Existence*, 584–593.

Choi, W., Pantofaru, C., and Savarese, S. (2011). Detecting and Tracking People using an RGB-D Camera via Multiple Detector Fusion. *IEEE International Conference on Computer Vision*, 1076–1083.

Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, 886–893.

Darrell, T., Demirdjian, D., Checka, N., and Felzenszwalb, P. (2001). Plan-view trajectory estimation with dense stereo background models. *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, *2*(February), 628–635.

Darrell, T., Gordon, G., and Harville, M. (2000). Integrated Person Tracking Using Stereo, Color, and Pattern Detection. *International Journal of Computer Vision (IJVC)*, *37*(2), 175–185.

Davies, N., Langheinrich, M., Jose, R., and Schmidt, A. (2012). Open Display Networks: A Communications Medium for the 21st Century. *Computer*, *45*(5), 58–64.

Davies, Nigel, Langheinrich, Marc, Krüger, A. (2016). Pervasive Displays. *IEEE Pervasive Computing*, *15*(3), 11–13.

De Boer, F. A. and Verbeek, F. J. (2009). Seeing through virtual windows. In *ACM Conference on Human Factors in Computing Systems*.

Dinh, T. B., Vo, N., and Medioni, G. (2011). Context tracker: Exploring supporters and distracters in unconstrained environments. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1177–1184.

D'Orazio, T., Leo, M., Mosca, N., Spagnolo, P., and Mazzeo, P. L. (2009). A semi-automatic system for ground truth generation of soccer video sequences. *6th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2009*, 559–564.

Du, W. and Piater, J. (2007). Multi-camera people tracking by collaborative particle filters and principal axis-based integration. In *Proc. of Asian Conference on Computer Vision*, 365–374. Springer Berlin Heidelberg.

Dundas, J. and Wagner, M. (2014). Adaptive Projection Displays: a low cost system for public interactivity. In *WSCG 2014: Poster Papers Proceedings: 22nd International Conference in Central Europeon Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS Association*, 55–60.

Elgammal, A., Harwood, D., and Davis, L. (2000). Non-parametric model for background subtraction. In *Computer VisionECCV*, 751–767.

Englebienne, G., Van Oosterhout, T., and Kröse, B. (2009). Tracking in sparse multi-camera setups using stereo vision. *2009 3rd ACM/IEEE International Conference on Distributed Smart Cameras, ICDSC 2009*.

Eveland, C., Konolige, K., and Bolles, R. C. (1998). Background modeling for segmentation of video-rate stereo sequences. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 266–271.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 3354–3361.

Gkalelis, N., Kim, H., Hilton, A., Nikolaidis, N., and Pitas, I. (2009). The i3DPost multi-view and 3D human action/interaction database. *CVMP 2009 - The 6th European Conference for Visual Media Production*, 159–168.

Gokturk, S. B., Yalcin, H., and Bamji, C. (2004). A Time-Of-Flight Depth Sensor System Description , Issues and Solutions. In *Computer Vision and Pattern Recognition Workshop*, 35 – 35.

Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEEE Proceedings of Radar and Signal Processing*, *140*(2), 107–113.

Han, J., Shao, L., Xu, D., and Shotton, J. (2013). Enhanced computer vision with Microsoft Kinect sensor: A review. *IEEE Transactions on Cybernetics*, *43*(5), 1318–1334.

Hare, S., Saffari, A., and Torr, P. H. S. (2011). Struck: Structured output tracking with kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(10), 2096–2109.

Harguess, J., Hu, C., and Aggarwal, J. K. (2011). Occlusion Robust Multi-Camera Face Tracking. In *IEEE Computer Society Conference, Computer Vision and Pattern Recognition Workshops (CVPRW), 2011*, 31–38.

Hartley, R. I. (1997). In Defence of the 8-point Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(6), 580–593.

Harville, M. (2004). Stereo person tracking with adaptive plan-view templates of height and occupancy statistics. *Image and Vision Computing*, *22*(2), 127–142.

Harville, M. and Dalong Li (2004). Fast, integrated person tracking and activity recognition with plan-view templates from a single stereo camera. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, *2*(July), 398–405.

Isard, M. and Blake, A. (1998). CONDENSATION Conditional Density Propagation for Visual Tracking. *Advances in Neural Information Processing Systems*, 361–367.

Johnsen, K. J., Wins, P., and Johnsen, K. (2013). Single Viewer Walk-around Quasi Volumetric Display. In *IEEE Workshop on Off-the-Shelf Virtual Reality.*

Kalal, Z. (2011). *Tracking Learning Detection*. Ph. D. thesis, University of Surrey.

Kalal, Z., Matas, J., and Mikolajczyk, K. (2008). Weighted Sampling for Large-Scale Boosting. *Procedings of the British Machine Vision Conference 2008*, 42.1–42.10.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2010a). Face-TLD: Tracking-Learning-Detection applied to faces. *Image Processing (ICIP), 2010 17th IEEE International Conference on. IEEE*, 3789–3792.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2010b). Forward-Backward Error: Automatic Detection of Tracking Failures. *2010 20th International Conference on Pattern Recognition*, 2756–2759.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2011). Tracking-Learning-Detection. *IEEE transactions on pattern analysis and machine intelligence*, *6*(1), 1–14.

Kalarot, R., Gimel'farb, G., and Morris, J. (2012). 3D object tracking with a high-resolution GPU based real-time stereo. *Proceedings of the 27th Conference on Image and Vision Computing New Zealand - IVCNZ '12*, 394–399.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, *82*(1), 35–45.

Kavasidis, I., Palazzo, S., Di Salvo, R., Giordano, D., and Spampinato, C. (2012). A semi-automatic tool for detection and tracking ground truth generation in videos. *Proceedings of the 1st International Workshop on Visual Interfaces for Ground Truth Collection in Computer Vision Applications - VIGTA '12*, 1–5.

Kavasidis, I., Palazzo, S., Salvo, R. D., Giordano, D., and Spampinato, C. (2014). An innovative web-based collaborative platform for video annotation. *Multimedia Tools and Applications*, *70*(1), 413–432.

Kavasidis, I., Spampinato, C., and Giordano, D. (2013). Generation of ground truth for object detection while playing an online game: Productive gaming or recreational working? *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 694–699.

Khan, S. M. and Shah, M. (2006). A Multiview Approach to Tracking People in Crowded Scenes Using a Planar Homography Constraint. In *European Conference on Computer Vision*, Volume 3954, 133–146.

Kim, M., Kumar, S., Pavlovic, V., and Rowley, H. (2008). Face Tracking and recognition with Visual Constrains in Real-world Videos. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–8.

Kobayashi, Y., Sugimura, D., Hirasawa, K., Suzuki, N., Kage, H., Sato, Y., and Sugimoto, A. (2006). 3D Head Tracking using the Particle Filter with Cascaded Classifiers. *Procedings of the British Machine Vision Conference 2006*, 5.1–5.10.

Kovacevic, J., Juric-Kavelj, S., Petrovic, I., Kovačević, J., and Petrović, I. (2011). An Improved CamShift Algorithm Using Stereo Vision For Object Tracking. *MIPRO, 2011 Proceedings of the 34th International Convention*, 707–710.

Kristan, M., Kovacic, S., Leonardis, A., and Pers, J. (2010). A two-stage dynamic model for visual tracking. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, *40*(6), 1505–1520.

Kristan, M., Matas, J., Leonardis, A., Vojir, T., Pflugfelder, R., Fernandez, G., Nebehay, G., Porikli, F., and Cehovin, L. (2016). A Novel Performance Evaluation Methodology for Single-Target Trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(11), 2137–2155.

Kristan, M., Pflugfelder, R., Leonardis, A., Matas, J., Čehovin, L., Nebehay, G., Vojívr, T., Fernandez, G., and Others (2015). The Visual Object Tracking VOT2015 Challenge Results. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, 564–586. IEEE.

Kwon, J. and Lee, K. M. (2009). Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive basin hopping monte carlo sampling. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, 1208–1215.

Lee, C., Kwon, Y.-m., and Kim, S.-k. (2013). Multi-User Pupil Tracking for 3D Multiview Display. In *Proceedings of 3D Systems and Applications (3DSA)*, 3–6.

Lee, J. C. (2008). Hacking the Nintendo Wii remote. *IEEE Pervasive Computing*, *7*(3), 39–45.

Li, A. M., Park, P. S., and Chen, Y. H. (2012). An adaptive particle filter tracking method based on homography and common FOV. In *Proceedings of the 2012 ACM*

Research in Applied Computation Symposium, New York, New York, USA, 126. ACM Press.

Li, T., Yuan, G., and Li, W. (2016). Particle filter with novel nonlinear error model for miniature gyroscope-based measurement while drilling navigation. *Sensors*, *16*(3), 371.

Li, X., Hu, W., Shen, C., Zhang, Z., Dick, A., and Hengel, A. V. D. (2013). A Survey of Appearance Models in Visual Object Tracking. *ACM transactions on Intelligent Systems and Technology (TIST)*, *4*(4), 58.

Li, Y., Ai, H., Yamashita, T., Lao, S., and Kawade, M. (2008). Tracking in low frame rate video: a cascade particle filter with discriminative observers of different life spans. *IEEE transactions on pattern analysis and machine intelligence*, *30*(10), 1728–40.

Liao, S., Jain, A. K., and Li, S. Z. (2016). A Fast and Accurate Unconstrained Face Detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(2), 211–223.

Llewellyn, G., Morgan, K., Member, S., Liu, J. G., and Yan, H. (2010). Precise Subpixel Disparity Measurement From Very Narrow Baseline Stereo. *IEEE Transactions on Geoscience and Remote Sensing*, *48*(9), 3424–3433.

Luber, M., Spinello, L., and Arras, K. O. (2011). People Tracking in RGB-D Data With On-line Boosted Target Models. In *2011 IEEE International Conference on Intelligent Robots and Systems*, 3844–3849.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging Understanding Workshop*, 121–130.

Maggio, E. (2007). Adaptive multifeature tracking in a particle filtering framework. *IEEE Transactions on Circuits and Systems for Video Technology*, *17*(10), 1348–1359.

Maimone, A. and Fuchs, H. (2012). Reducing Interference Between Multiple Structured Light Depth Sensors Using Motion. In *Virtual Reality Short Papers and Posters (VRW), IEEE*, 51–54.

Malleson, C. and Collomosse, J. (2013). Virtual Volumetric Graphics on Commodity Displays Using 3D Viewer Tracking. *International Journal of Computer Vision*, *101*, 519–532.

Mastorakis, G. and Makris, D. (2014). Fall detection system using Kinect's infrared sensor. *Journal of Real-Time Image Processing*, *9*(4), 635–646.

Mathias, M., Benenson, R., Pedersoli, M., and Van Gool, L. (2014). Face detection without bells and whistles. In *European Conference on Computer Vision. Springer*, 720–735.

Mihalcik, D. and Doermann, D. (2003). The Design and Implementation of ViPER. In *University of Maryland*, 234–241.

Mikhisor, M., Wyvill, G., Mccane, B., and Mills, S. (2014). 3D Face Tracking in Fisheye Stereo Video Using Particle Filters. In *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand. ACM, 2014.*

Mikhisor, M., Wyvill, G., McCane, B., and Mills, S. (2015). Adapting Generic Trackers for Tracking Faces. In *Proc. of Image and Vision Computing New Zealand (IVCNZ), 2015 International Conference, IEEE*, 1–6.

Mittal, A. and Davis, L. S. (2002). M2Tracker: A Multi-View Approach to Segmenting and Tracking People in a Cluttered Scene Using Region-Based Stereo. In *European conference on computer vision. Springer Berlin Heidelberg*, 18–33.

Mittal, A. and Davis, L. S. (2003). M2 tracker: A multi-view approach to segmenting and tracking people in a cluttered scene. *International Journal of Computer Vision*, *51*(3), 189–203.

Muller, J., Wilmsmann, D., Exeler, J., Buzeck, M., Schmidt, A., Jay, T., and Kruger, A. (2009). Display Blindness: The Effect of Expectations on Attention towards Digital Signage. In *Pervasive Computing*, 1–8.

Muñoz-Salinas, R. (2007). People Detection and Tracking Using Stereo Vision and Color. *Image and Vision Computing*, *25*(6), 995–1007.

Muñoz-Salinas, R. (2008). A Bayesian plan-view map based approach for multiple-person detection and tracking. *Pattern Recognition*, *41*(12), 3665–3676.

Muñoz-Salinas, R., Aguirre, E., and García-Silvente, M. (2007). People detection and tracking using stereo vision and color. *Image and Vision Computing*, *25*(6), 995–1007.

Muñoz-Salinas, R., García-Silvente, M., and Medina Carnicer, R. (2008). Adaptive multi-modal stereo people tracking without background modelling. *Journal of Visual Communication and Image Representation*, *19*(2), 75–91.

Muscoloni, A. and Mattoccia, S. (2014). Real-time tracking with an embedded 3D camera with FPGA processing. *International Conference on 3D Imaging (IC3D), 2014*, 1–7.

Nakanishi, Y., Fujii, T., and Kiatjima, K. (2002). Vision-Based Face Tracking System for Large Displays. *Proceedings of The International Conference on Ubiquitous Computing (UbiComp '02)*, 152–159.

Nanda, H. and Fujimura, K. (2004). Visual Tracking Using Depth Data. In *Conference on Computer Vision and Pattern Recognition Workshop, 2004*, 37–37.

Nebehay, G. (2012). Robust Object Tracking Based on Tracking-Learning-Detection. Technical report, Technischen Universität Wien.

Nickel, K. and Gehrig, T. (2005). A joint particle filter for audio-visual speaker tracking. *Proc. of International Conference on Multimodal Interfaces*, 61–68.

Nummiaro, K., Koller-Meier, E., and Svoboda, T. (2003). Color-based object tracking in multi-camera environments. *Pattern Recognition, Springer Berlin Heidelberg*, 591–599.

Oygard, M. A. (2012). Head Tracking With WebRTC.

Pantic, M. and Bartlett, M. S. (2007). Machine analysis of facial expressions. In *Face recognition, InTech*.

Park, U., Choi, H. C., Jain, A. K., and Lee, S. W. (2013). Face tracking and recognition at a distance: A coaxial and concentric ptz camera system. *IEEE Transactions on Information Forensics and Security*, *8*(10), 1665–1677.

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep Face Recognition. In *Procedings of the British Machine Vision Conference 2015*, Number Section 3, 41.1–41.12.

Pérez, P., Hue, C., Vermaak, J., and Gangnet, M. (2002). Color-Based Probabilistic Tracking. In *Proc. of European Conference on Computer Vision*, 661–675.

Pham, H. X. and Pavlovic, V. (2016). Robust real-time 3D face tracking from RGBD videos under extreme pose, depth, and expression variation. *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, 441–449.

Pulli, K., Baksheev, A., Kornyakov, K., and Eruhimov, V. (2012). Real-time computer vision with OpenCV. *Communications of the ACM*, *55*(6), 61.

Radke, R., Andra, S., Al-Kofahi, O., and Roysam, B. (2005). Image change detection algorithms: a systematic survey. *IEEE Transactions on Image Processing*, *14*(3), 294–307.

Rautaray, S. S. and Agrawal, A. (2015). Vision based hand gesture recognition for human computer interaction: a survey. *Artificial Intelligence Review*, *43*(1), 1–54.

Rekimoto, J. and Building, T. M. (1995). A Vision-Based Head Tracker for Fish Tank Virtual Reality - VR without Head Gear -. In *Virtual Reality Annual International Symposium, 1995. Proceedings. IEEE*, 94–100.

Ren, G., Li, C., O'Neill, E., and Willis, P. (2013). 3D freehand gestural navigation for interactive public displays. *IEEE Computer Graphics and Applications*, *33*(2), 47–55.

Rodgers, J. L. and Nicewander, W. A. (1988). Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician*, *42*(1), 59 – 66.

Ross, D., Lim, J., Lin, R.-S., and Yang, M.-H. (2008). Incremental Learning for Robust Visual Tracking. *International Journal of Computer Vision*, *77*(1-3), 125–141.

Rougier, C., Auvinet, E., Rousseau, J., Mignotte, M., and Meunier, J. (2011). Fall Detection from Depth Map Video Sequences. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Volume 6719, 121–128.

Rougier, C. and Meunier, J. (2010a). 3D Head Trajectory Using a Single Camera. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Volume 6134 LNCS, 505–512.

Rougier, C. and Meunier, J. (2010b). 3D head trajectory using a single camera. *International Journal of Future Generation Communication and Networking*, *3*(4), 505–512.

Salti, S., Cavallaro, A., and Di Stefano, L. (2012). Adaptive appearance modeling for video tracking: Survey and evaluation. *IEEE Transactions on Image Processing*, *21*(10), 4334–4348.

Samarawickrama, M. G. (2010). *Performance Evaluation of Vision Algorithms on FPGA*. Ph. D. thesis, University of Moratuwa.

Saragih, J. M., Lucey, S., and Cohn, J. F. (2011). Deformable Model Fitting by Regularized Landmark Mean-Shift. *International Journal of Computer Vision*, *91*(2), 200–215.

Scaramuzza, D., Martinelli, a., and Siegwart, R. (2006). A Toolbox for Easily Calibrating Omnidirectional Cameras. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (2006)*, (June), 5695–5701.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 815–823.

Sigal, L., Balan, A. O., and Black, M. J. (2010). 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.

Sippl, A., Holzmann, C., Zachhuber, D., and Ferscha, A. (2010). Real-time gaze tracking for public displays. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6439 LNCS*(818652), 167–176.

Smeulders, A. W. M., Chu, D. M., Cucchiara, R., Calderara, S., Dehghan, A., and Shah, M. (2014). Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(7), 1442–1468.

Stone, E. E. and Skubic, M. (2014). Fall Detection in Homes of Older Adults Using the Microsoft Kinect. *IEEE journal of biomedical and health informatics*, *19*(1), 290–301.

Sudowe, P. and Leibe, B. (2011). Efficient use of geometric constraints for sliding-window object detection in video. *Proceedings of the 8th international conference on Computer vision systems*, 11–20.

Sun, Y., Wang, X., and Tang, X. (2014). Deep Learning Face Representation From Predicting 10 000 Classes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1891–1898.

Sunderrajan, S. and Manjunath, B. (2013). Multiple view discriminative appearance modeling with IMCMC for distributed tracking. *2013 Seventh International Conference on Distributed Smart Cameras (ICDSC)*, 1–7.

Surman, P., Day, S., Liu, X., Benjamin, J., Urey, H., and Aksit, K. (2015). Head tracked retroreflecting 3D display. *Journal of the Society for Information Display*, *23*(2), 56–68.

Taigman, Y., Yang, M., and Ranzato, M. (2014). Deepface: Closing the gap to human -level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1701–1708.

Tang, F., Tao, H., Harville, M., and Robinson, I. N. (2008). Fusion of local appearance with stereo depth for object tracking. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, 1–8.

Tao Zhao, Aggarwal, M., Kumar, R., and Sawhney, H. (2005). Real-Time Wide Area Multi-Camera Stereo Tracking. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, *1*, 976–983.

Tippetts, B., Lee, D. J., Lillywhite, K., and Archibald, J. (2016). Review of stereo vision algorithms and their suitability for resource-limited systems. *Journal of Real-Time Image Processing*, *11*(1), 5–25.

Tulyakov, S., Vieriu, R.-L., Sangineto, E., and Sebe, N. (2015). FaceCept3D: Real Time 3D Face Tracking and Analysis. *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, 28–33.

Turaga, P., Member, S., Chellappa, R., and Subrahmanian, V. S. (2008). Machine Recognition of Human Activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, *18*(11), 1473–1488.

Viola, P. and Jones, M. (2001). Robust real-time face detection. *Proce. of 8th IEEE International Conference on Computer Vision*, *2*(2), 137–154.

Vondrick, C., Patterson, D., and Ramanan, D. (2013). Efficiently scaling up crowdsourced video annotation: A set of best practices for high quality, economical video labeling. *International Journal of Computer Vision*, *101*(1), 184–204.

Wai, A. W. Y., Tahir, S. M., and Chang, Y. C. (2015). GPU acceleration of real time Viola-Jones face detection. In *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 183–188. IEEE.

Wang, J., Yin, L., Wei, X., and Sun, Y. (2006). 3D Facial Expression Recognition Based on Primitive Surface Feature Distribution. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference*, 1399–1406.

Wang, L., Hu, W., and Tan, T. (2003). Recent developments in human motion analysis. *Pattern recognition*, *36*(3), 585–601.

Wang, Y.-D., Wu, J.-K., and Kassim, A. A. (2005). Particle Filter for Visual Tracking Using Multiple Cameras. *MVA2005 IAPR Conference on Machine VIsion Applications*.

Wei Zhong, Huchuan Lu, and Ming-Hsuan Yang (2012). Robust object tracking via sparsity-based collaborative model. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1838–1845. IEEE.

Wu, Y., Lim, J., and Yang, M. H. (2013). Online object tracking: A benchmark. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2411–2418.

Wu, Y., Lim, J., and Yang, M.-H. (2015). Object Tracking Benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *37*(9), 1834–1848.

Xiao, J., Kanade, T., and Cohn, J. F. (2002). Robust full-motion recovery of head by dynamic templates and re-registration techniques. *Proceedings - 5th IEEE International Conference on Automatic Face Gesture Recognition, FGR 2002*, 163–169.

Xiaoyu Huang, Liyuan Li, and Sim, T. (2004). Stereo-based human read detection from crowd scenes. In *2004 International Conference on Image Processing, 2004. ICIP '04.*, Volume 2, 1353–1356. IEEE.

Xu Jia, Huchuan Lu, and Ming-Hsuan Yang (2012). Visual tracking via adaptive structural local sparse appearance model. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1822–1829. IEEE.

Xue, Y. and Wang, Y. (2014). Multi-User Autostereoscopic 2D/3D Switchable Flat-Panel Display. *Journal of Display Technology*, *10*(9), 737–745.

Yang, H., Shao, L., Zheng, F., Wang, L., and Song, Z. (2011). Recent advances and trends in visual tracking: A review. *Neurocomputing*, *74*(18), 3823–3831.

Yebes, J. J., Bergasa, L. M., Arroyo, R., and Lázaro, A. (2014). Supervised learning and evaluation of KITTI's cars detector with DPM. In *Intelligent Vehicles Symposium Proceedings*, 768–773.

Yilmaz, A. and Javed, O. (2006). Object Tracking: A Survey. *ACM Computing Surveys (CSUR)*, *38*(4), 13.

Yuen, J., Russell, B., Ce Liu, and Torralba, A. (2009). LabelMe video: Building a video database with human annotations. In *2009 IEEE 12th International Conference on Computer Vision*, 1451–1458.

Zhang, S., Gong, Y., Huang, J. B., Lim, J., Wang, J., Ahuja, N., and Yang, M. H. (2016). Tracking persons-of-interest via adaptive discriminative features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9909 LNCS*(August), 415–433.

Zhang, Y., Bulling, A., and Gellersen, H. (2013). SideWays: A Gaze Interface for Spontaneous Interaction with Displays. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 851.

Zhang, Y., Chong, M. K., Müller, J., Bulling, A., and Gellersen, H. (2015). Eye tracking for public displays in the wild. *Personal and Ubiquitous Computing*, *19*(5-6), 967–981.

Zhang, Z. (2000). A Flexible New Technique for Camera Calibration. *IEEE Transactions on pattern analysis and machine intelligence*, *22*(11), 1330–1334.

Zoidi, O., Nikolaidis, N., and Pitas, I. (2013). Appearance based object tracking in stereo sequences. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2434–2438.

Zoidi, O., Nikolaidis, N., Tefas, A., and Pitas, I. (2014). Stereo object tracking with fusion of texture, color and disparity information. *Signal Processing: Image Communication*, *29*(5), 573–589.