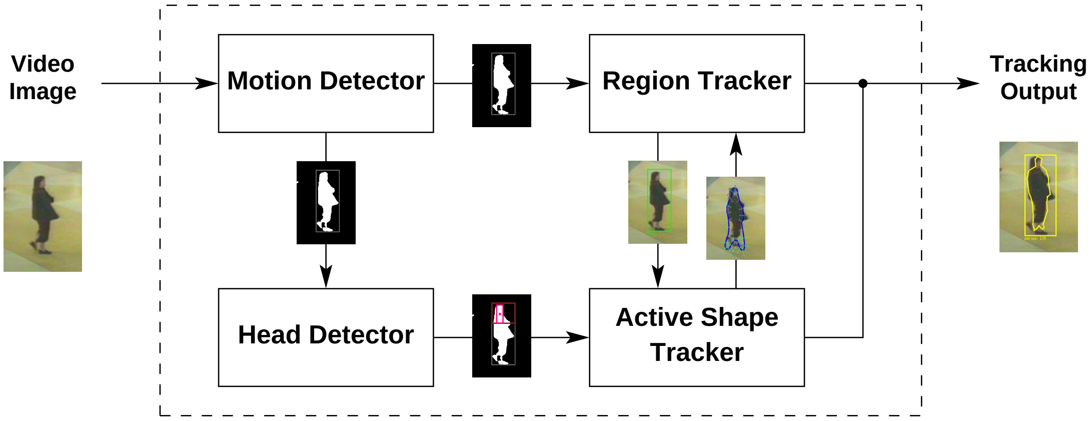
The University of Reading

Design and Implementation of People Tracking Algorithms for Visual Surveillance Applications

Nils T Siebel

Computational Vision Group
Department of Computer Science


Video Image
Motion Detector
Region Tracker
Tracking Output
Head Detector
Active Shape Tracker

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

March 2003

For my parents.

Copyright Notice

The original material presented in this thesis is protected by copyright. The copyright is owned by the author, © Nils T Siebel, 2003. The research work and results may be used for any non-military, academic research purpose if the source is adequately acknowledged. Any commercial exploitation of the ideas or other material presented here requires the prior written permission of the author. No part of the work described here may ever be used for any military purpose, research or not.

The source code of the Reading People Tracker introduced in this thesis has been published under the GNU Public License (GPL) and is available through the URI `http://www.cvg.cs.rdg.ac.uk/~nts/PeopleTracking/`.

Abstract

This thesis describes the design and implementation of a people tracking module for an integrated visual surveillance system developed as part of the Framework V project `ADVISOR`, funded by the European Union. Starting from an earlier people tracker developed at the Universities of Leeds and Reading over the past 10 years, appropriate methods have been included and adapted to meet the special needs which exist in visual surveillance systems. The result is the *Reading People Tracker*.

The emphasis in this work lies on the completion of the whole task: from image processing and tracking algorithms through to system design and integration.

The *Reading People Tracker* uses a combination of four co-operating detection and tracking modules to track people in camera images. Each of the modules is of medium to low complexity to ensure realtime performance. The modules are a Motion Detector, a Region Tracker, a Head Detector and an Active Shape Tracker. Combining different modules and letting them exchange their results overcomes the limitations which individual modules have and results in a greater reliability. All four modules work directly with the image in order to reduce the complexity of the tracker. Complicated 3D models are avoided. Experiments show that the *Reading People Tracker* tracks individuals robustly in the presence of occlusion and low image quality.

An important aspect of the *Reading People Tracker* is its maintainability. The software was re-engineered, making it possible to adapt, extend and improve the software with ease. This re-engineering process and its influence on the maintainability of the people tracker is analysed as a case study.

A second aspect examined in the case study is the software processes which have been used to maintain the software over its lifetime of 10 years. It is shown that the diversity of software processes has a significant impact on the quality of the software. Process diversity is divided into two broad classes, *latitudinal* and *longitudinal process diversity*. The influence of both classes of process diversity on software quality is examined.

Declarations

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Part of the work in this thesis has been presented in the following publications:

Computer Vision Publications

[1] Nils T Siebel and Steve Maybank. Fusion of multiple tracking algorithms for robust people tracking. In Anders Heyden, Gunnar Sparr, Mads Nielsen and Peter Johansen, editors, *Proceedings of the 7th European Conference on Computer Vision (ECCV 2002), København, Denmark*, volume IV, pages 373–387, May 2002.

[2] Nils T Siebel and Steve Maybank. Real-time tracking of pedestrians and vehicles. In *Proceedings of the 2nd IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS'2001), Kauai, USA*, December 2001. CD-ROM proceedings, pages unnumbered.

[3] Nils T Siebel and Steve Maybank. On the use of colour filtering in an integrated real-time people tracking system. In Paolo Remagnino, Graeme A Jones, Nikos Paragios and Carlo Regazzoni, editors, *Video Based Surveillance Systems: Computer Vision and Distributed Processing*, chapter 14, pages 167–175. Kluwer Academic Publishers, Boston, USA, 2001.

[4] Nils T Siebel and Steve Maybank. The application of colour filtering to real-time person tracking. In *Proceedings of the 2nd European Workshop on Advanced Video-Based Surveillance Systems (AVBS'2001), Kingston upon Thames, UK*, pages 227–234, September 2001.

Software Engineering Publications

[5] Nils T Siebel, Steve Cook, Manoranjan Satpathy and Daniel Rodríguez. Latitudinal and longitudinal process diversity. *Journal of Software Maintenance and Evolution*, 15(1):9–25, January–February 2003.

[6] Manoranjan Satpathy, Nils T Siebel and Daniel Rodríguez. Maintenance of object oriented systems through re-engineering: A case study. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002), Montréal, Canada*, pages 540–549, October 2002.

Manual of the Reading People Tracker

[7] Tugdual Le Bouffant, Nils T Siebel, Stephen Cook and Steve Maybank. The Reading People Tracker Version 1.12 Reference Manual. Technical Report RUCS/2002/TR/11/001/A, Department of Computer Science, The University of Reading, Reading, UK, November 2002.

Contents

Copyright Notice **v**

Abstract **vii**

Declarations **ix**

Contents **xi**

1 Introduction **1**
- 1.1 People Tracking for Visual Surveillance . . . 1
- 1.2 Issues Addressed in this Thesis . . . 2
  - 1.2.1 New People Tracking Algorithms . . . 3
  - 1.2.2 Software Engineering Aspects . . . 3
- 1.3 Overview of the Thesis . . . 4
- 1.4 Acknowledgements . . . 5

2 People Tracking **7**
- 2.1 Concepts and Terminology . . . 7
  - 2.1.1 Two Basic Definitions . . . 7
  - 2.1.2 Motion Detection . . . 7
  - 2.1.3 Modelling . . . 9
- 2.2 Related Work . . . 11
  - 2.2.1 Overview and Classification . . . 11
  - 2.2.2 Baumberg's *Leeds People Tracker* . . . 12
  - 2.2.3 Haritaoglu's W^4 System . . . 16
  - 2.2.4 Gavrila's 3-D Model-based People Tracker . . . 19
  - 2.2.5 Sidenbladh's 3D People Tracker . . . 21
  - 2.2.6 Conclusions . . . 23
- 2.3 Building an Integrated People Tracking Application . . . 23
  - 2.3.1 The `ADVISOR` Project and its Objectives . . . 23
  - 2.3.2 System Overview . . . 24

2.3.3 Development Plan . . . 26
2.3.4 Data Formats for Communication . . . 27
2.3.5 Building a People Tracking Module for `ADVISOR` . . . 28

3 Colour Image Filtering for Robust Image Processing 31
3.1 Introduction . . . 31
3.1.1 The Representation of Colour in Computers . . . 31
3.1.2 Image Generation and Transmission . . . 33
3.1.3 Determining the Effects of Image Noise . . . 35
3.2 Image Noise and Image Filtering . . . 35
3.2.1 Use of Images Within the People Tracker . . . 35
3.2.2 Filtering Techniques: Creating the Difference Image . . . 37
3.3 Experiments and Results . . . 39
3.3.1 The Test Data . . . 39
3.3.2 Local Image Filters Examined Here . . . 40
3.3.3 Method a: Filtering the Difference Image . . . 42
3.3.4 Method b: Differencing the Filtered Images . . . 44
3.4 Discussion . . . 46

4 The Reading People Tracker 47
4.1 Designing a People Tracker for `ADVISOR` . . . 47
4.2 The Tracking Algorithm: Overview and Structure . . . 48
4.2.1 Overview and General Features . . . 48
4.2.2 Module 1: The Motion Detector . . . 50
4.2.3 Module 2: The Region Tracker . . . 50
4.2.4 Module 3: The Head Detector . . . 54
4.2.5 Module 4: The Active Shape Tracker . . . 56
4.3 Module Interaction and Other Features . . . 58
4.3.1 Interaction Between Tracking Modules . . . 58
4.3.2 Hypothesis Refinement . . . 61
4.3.3 Software Engineering Aspects . . . 61
4.4 Demonstration and Discussion . . . 62
4.4.1 Testing on a London Underground Sequence . . . 62
4.4.2 Analysis . . . 64
4.4.3 Remaining Problems . . . 66
4.4.4 Summary . . . 67

5 Maintainability of the Reading People Tracker 69
5.1 Introduction . . . 69
5.1.1 Software Maintenance Techniques . . . 69
5.1.2 Related Work . . . 71

5.2 The People Tracker and its Maintainability 72
5.2.1 Brief History 73
5.2.2 Motivation for Re-design 74
5.3 Approaches Taken for the Re-design 75
5.3.1 Stages of Work 75
5.3.2 Maintenance Effort 77
5.3.3 Code Size 78
5.4 Further Analysis and Discussion 78
5.4.1 Generated Artefacts of the People Tracker 78
5.4.2 Experience with the Re-engineered software 79
5.4.3 Improved Maintainability 80
5.4.4 Personnel Factors 81
5.4.5 Lessons Learned 82
5.5 Conclusions 84

6 Process Diversity and its Implications for the People Tracker 85
6.1 Introduction 85
6.1.1 Concepts and Management of Process Diversity 86
6.1.2 Related Work 86
6.2 Process Diversity 87
6.2.1 Latitudinal Process Diversity 88
6.2.2 Longitudinal Process Diversity 89
6.2.3 Evolution-oriented Models of Software Processes 90
6.3 Description of the People Tracker Software 93
6.3.1 Brief History 94
6.3.2 Current Status and Outlook 95
6.4 Analysis of the Process Diversity for the People Tracker 96
6.4.1 Latitudinal Process Diversity: Diversity between Co-operating Groups 96
6.4.2 Longitudinal Process Diversity and Transition between Process Models 97
6.4.3 Metrics and Further Analysis 99
6.5 Lessons Learned 100
6.5.1 Latitudinal Process Diversity 101
6.5.2 Longitudinal Process Diversity 101
6.6 Conclusions 102

7 Validation of the New People Tracker 105
7.1 Experimental Setup 105
7.1.1 Test Data 105
7.1.2 Configuration of the People Tracker 106

7.2 Experimental Results 107
7.3 Summary and Conclusions 111
7.3.1 Tracking Performance 111
7.3.2 Processing Speed 113
7.3.3 Conclusions 113

8 Conclusions 115
8.1 Summary of Work 115
8.2 Discussion 116
8.3 Future Work 116

List of Figures 119

List of Tables 121

Bibliography 123

Index 133

Chapter 1

Introduction

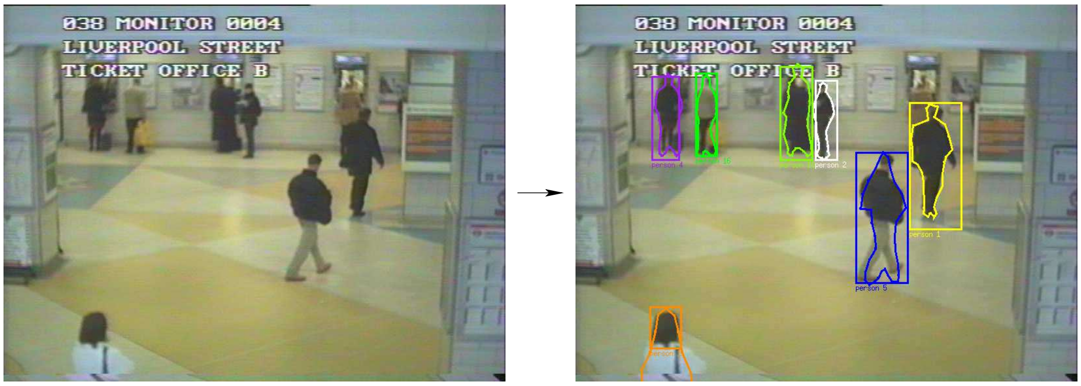
1.1 People Tracking for Visual Surveillance

With recent advances of computer technology automated visual surveillance has become a popular area for research and development. Surveillance cameras are installed in many public areas to improve safety, and computer-based image processing is a promising means to handle the vast amount of image data generated by large networks of cameras. A number of algorithms to track people in camera images can be found in the literature, yet so far little research has gone into building realtime visual surveillance systems that integrate such a people tracker.

The task of an integrated surveillance system is to warn an operator when it detects events which may require human intervention, for example to avoid possible accidents or vandalism. These warnings can only be reliable if the system can detect and understand human behaviour, and for this it must locate and track people reliably. Figure 1.1 illustrates this task. Image 1.1(a) has been taken by a surveillance camera overlooking a ticket office in a London Underground station. Each visible person in the image has to be detected and given a unique label as in Figure 1.1(b). A good people tracker always assigns the same label to the same person and thereby makes long-term tracking information available for subsequent behaviour analysis.

Figure 1.1 also shows some of the difficulties involved in this people tracking task:

- People can occlude each other in the image. e.g. person 1 (marked in yellow) occludes the person at the rightmost ticket office counter.
- People can have a low contrast to the background so that they cannot be picked up easily by the system.
- The image quality and lighting conditions (visibility) in the station can be bad.



(a) Image from an underground station

(b) Identified and tracked people

Figure 1.1: The task of detecting and tracking people

1.2 Issues Addressed in this Thesis

The research presented in this thesis examines the task of designing and implementing a people tracking module for a particular integrated visual surveillance system called `ADVISOR`. `ADVISOR` has been built by a European project also called `ADVISOR`[1] and involving three academic and three industrial partners. It was our group's task to build a people tracking subsystem for `ADVISOR`, and the results are presented in this thesis. The main focus is therefore on suitable **people tracking algorithms** and **software engineering aspects** connected with this task. Figure 1.2 shows an overview of the `ADVISOR` system and how the People Tracking subsystem is connected to other subsystems. More details on the `ADVISOR` project and system can be found in Section 2.3.1 below.

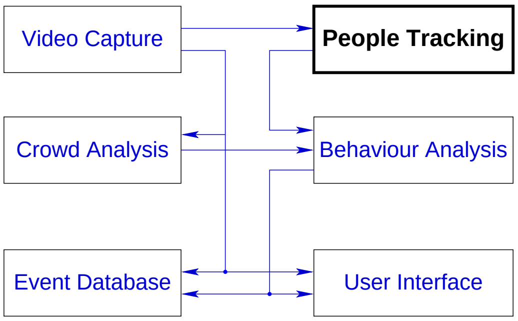


Figure 1.2: People Tracking as one of six subsystem of `ADVISOR`

---

[1] The project web site is located at `http://www-sop.inria.fr/orion/ADVISOR/`

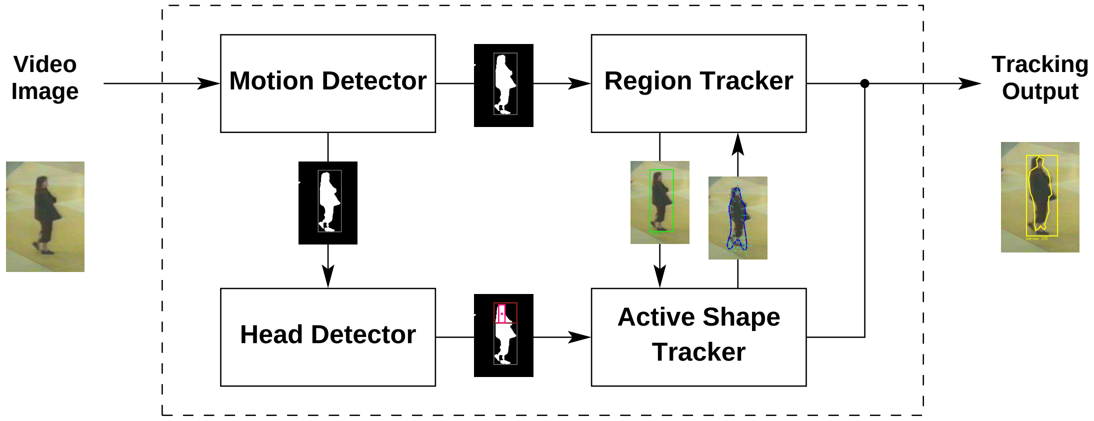


Figure 1.3: Structure of the *Reading People Tracker*

1.2.1 New People Tracking Algorithms

In this thesis a new people tracker, called the *Reading People Tracker*, has been built and validated. It is designed to track people in image sequences from surveillance cameras in underground stations. The Reading People Tracker is based on an existing people tracker, the *Leeds People Tracker* by Adam Baumberg (see Baumberg, 1995). This tracker has been extended and re-designed to improve tracking performance and meet new requirements.

The Reading People Tracker has the following functional features:

- It has been designed to run either as a subsystem of `ADVISOR` as shown in Figure 1.2 or as a standalone people tracker.
- It accepts input from multiple video feeds, processing data from all cameras in parallel.
- It comprises four detection and tracking modules which co-operate to create the overall tracking output. The modules aid each other to achieve a high level of tracking robustness and to overcome the limitations of individual modules. An overview of these modules and their interactions can be found in Figure 1.3.
- It has the ability to filter images in a way which minimises the influence of image noise and bad image quality on tracking algorithms.

1.2.2 Software Engineering Aspects

One important part of the Reading People Tracker developed for this thesis is an extension, in functionality and application, of the *Leeds People Tracker*. This means that a significant part of the software has been maintained for 10 years. During this time it has undergone major changes as its intended use and desired functionality changed with different projects and different people working on it. The negative effects of this on the maintainability of the code were significant, and in its original state it was not suitable for use within `ADVISOR`. Therefore the software was completely re-engineered, yielding a new design. The new software is highly maintainable and portable, and new functionality and tracking algorithms for `ADVISOR` have been added with ease.

The practical work on re-engineering the people tracker is supported by research into software maintainability and software processes. The main results of the research are the following:

- A case study of the Reading People Tracker and its recent history shows how a high level of maintainability was established through re-engineering and re-design.
- The processes used to build and maintain the people tracker over its life history have been examined. The influence of these different processes on the software has been established.
- Based on the examination of these processes a new classification scheme of process diversity into *latitudinal* and *longitudinal process diversity* has been devised and their influence on the software was determined.

1.3 Overview of the Thesis

Chapter 2 describes the task of people tracking and puts it into the context of the integrated surveillance system `ADVISOR`. An overview of the state-of-the-art in people tracking is given. The remaining chapters describe the original work of this thesis and are organised as follows.

Chapter 3 examines the use of **colour filtering techniques** to improve the robustness of image processing algorithms in the presence of the image noise that is typical for camera systems in underground stations.

Chapter 4 introduces the **Reading People Tracker** which was developed for this thesis, starting from existing people tracking software. Intended for automated visual surveillance, the algorithms are designed to operate reliably and in realtime. A key design aspect for the new people tracker is the combination of four co-operating detection and tracking modules to achieve robust tracking of pedestrians. A demonstration of the new people tracker and its key features is given.

Chapter 5 presents a case study of the people tracker which focuses on its **software maintainability**. It is shown how the software evolved over its lifetime of 10 years and how its maintainability was significantly improved using software re-engineering techniques.

Chapter 6 examines the influence of **software process diversity** on the people tracker over its lifetime. Inspired by this case study a new classification scheme for process diversity is presented. It distinguishes two broad types of process diversity, *latitudinal* and *longitudinal process diversity*. The influence of both types on the quality of software is examined.

Chapter 7 demonstrates the applicability of the new Reading People Tracker and its algorithms by **validating** it against test data collected from an underground station in Brussels.

Chapter 8 summarises and discusses the presented work.

1.4 Acknowledgements

First and foremost I wish to thank the people who gave me the chance to work in this interesting field of research. These are mostly my supervisor Steve Maybank and my previous teachers. I would not be here, and I could not have done this without my family and friends to whom I would like to express my utmost gratitude. Many thanks also to the many nice people in the Department and to our system administrators (especially Anthony Worrall) for helping to keep me and our computers happy.

This research was supported by the European Union, grant `ADVISOR` (IST-1999-11287), for which I am very grateful. I wish to thank London Underground Ltd, the Société des Transports Intercommunaux Bruxellois as well as our project partners Sergio Velastin (Kingston University), Michel Renard (Vigitec, Brussels) and the staff at Thales Research Ltd for providing video sequences, and our visitors and students for their valuable coding work. The co-operation of the other project partners (the ORION group at INRIA Sophia Antipolis, Bull S.A. Les Clayes-sous-Bois), and many useful discussions with these and other vision researchers are also gratefully acknowledged.

Many thanks to Hedvig Sidenbladh and Dariu Gavrila for letting me use their images in Section 2.2.

Chapter 2

People Tracking

In this chapter general People Tracking concepts are explained and an overview of the state-of-the-art in the area is presented.

2.1 Concepts and Terminology

2.1.1 Two Basic Definitions

In this thesis, a number of computer vision terms will be used which are either not commonly well-defined or which are sometimes used in different ways by different people. The following definitions specify how these terms are used.

2.1. Definition (BLOB)**:** *A blob is a connected set of pixels in an image. The pixels contained in a blob must be locally distinguishable from pixels which are not part of the blob.*

2.2. Definition (REGION)**:** *A region is the rectangular subimage defined by the minimal enclosing bounding box of an image area containing one or more blobs.*

Figure 2.1 shows an example image with three *blobs*, b_1, b_2 and b_3 and a *region* r which contains b_1 and b_2. Note that according to definition 2.2 a region could contain only b_1, or all three *blobs*.

2.1.2 Motion Detection

In order to detect moving people or other objects in the image, a motion detector normally maintains a model of the *background*, that is, the appearance of the scene without people in it. Any pixel in the current video image is then classified as “moving” if it cannot be explained by this model. This results in a binary *Motion*

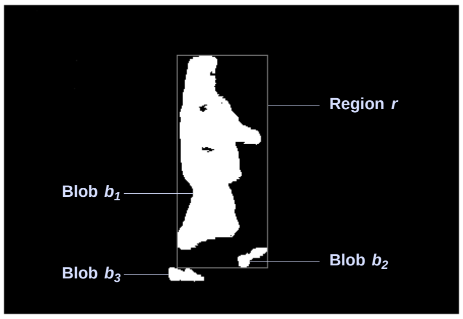


Figure 2.1: Example image with three blobs and a region r containing two of the blobs, b_1 and b_2.

Image with each pixel classified as either “moving” or “non-moving”. An example Motion Image is shown in Figure 2.2.

The motion detector obtains the *background model* by a process which usually involves processing the previous n video images, $n > 0$. One example for a background model is a pixelwise *median filter* of the video images over time. n is the *run-length* of the median filter.

There are a number of other background modelling methods. A very simple one is a constant image of the scene taken when there were no people in the image. In some situations such a simple method is not sufficient, e.g. in outdoor scenes where the lighting conditions change over time, and where the background itself can contain movement, as it happens with trees in the wind. More complex methods address these difficulties using elaborate statistical models for each pixel. Widely used examples are the multi-modal statistical motion detector from MIT which models each pixel with a mixture of 3 to 5 Gaussian distributions (Stauffer and Grimson, 1999; Stauffer et al., 1998) and the non-parametric statistical method from the University of Maryland (Elgammal et al., 2000). These methods have the disadvantage that they require a significant amount of computational time, which at present limits their use in realtime systems. Some methods (e.g. Rowe and Blake, 1996) try to overcome this by carrying out the computationally expensive background modelling process in an offline training stage. This means, however, that the system cannot adapt to changes in background or lighting conditions while it is operating.

The output from a motion detector is usually the *motion image* with classified pixels, like the one in Figure 2.2. Sometimes the output also includes a list of moving *blobs* extracted from this image, and the *regions* containing them.

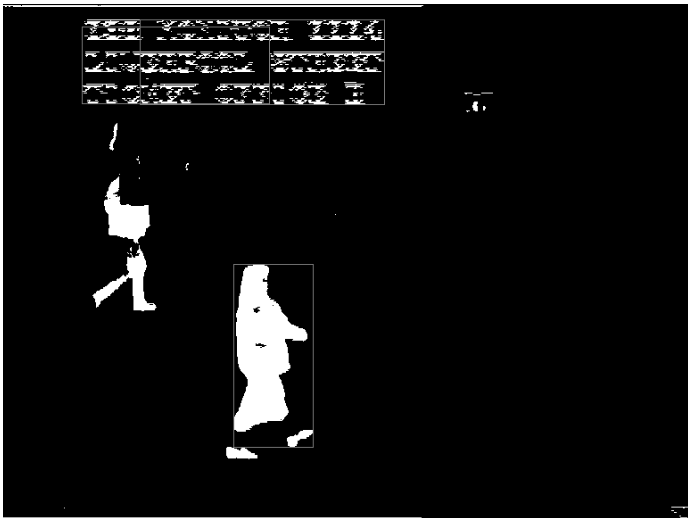


Figure 2.2: Example Motion Image

2.1.3 Modelling

If one is to distinguish people from other, possibly moving, objects in the image a certain notion of "person" or "non-person" has to be adopted. Usually, this is done by defining a *model* of a human being which directly or indirectly describes the appearance of a person in the image.

A *model* may be adjustable to the given data (e.g. image measurements) using *model parameters*. The following notion of *parameterised model* will be used in this thesis.

2.3. Definition (PARAMETERISED MODEL): *A parameterised model is a function m together with a parameter space P such that m maps* model parameters *$p \in P$ to a representation d of data: $d = m(p)$.*

The dimensionality of the parameter space P is called the *dimensionality* of the model m.

Model Complexity

One important issue when modelling the visual appearance of people or other objects is the complexity of the model. The two foremost problems are the facts that humans are complex articulated bodies, and their appearance changes depending on the camera view and on any occlusions.

Simple models of humans are usually easy to implement and fast enough to be used in realtime systems. However, they often lack the ability to adapt to the changing visual appearances of humans under changes of posture and viewing angle. These models might also have difficulties in the presence of occlusion.

More complex models have a greater ability to adapt. This makes them more general and more powerful, however, the increase in complexity has some disadvantages, too. Apart from being difficult to realise, their computational demands can make them unsuitable for realtime applications. Moreover, the high dimensionality of the "state" of the human model produces the need for some means to handle difficulties like visual ambiguities. For instance, if the tracker models the position and posture of a person's hand, this aspect might be undefined when the hand is occluded. Complex models are also more likely to fail in situations which are not expected by the system (for example, missing markers on objects, sitting people as opposed to walking people, change in lighting situations, moving background etc).

These issues make the process of choosing a suitable model an important issue in the design of a system. The trade-off between the advantages and disadvantages discussed above is usually decided by the demands of the respective application—such as speed and detail of required output.

Classification of Human Models

While the level of complexity in human modelling for tracking varies over a wide range, the methods can be classified into three main categories of increasing model complexity:

Category 1: Methods using region- or blob-based tracking, sometimes with additional classification schemes based on colour, texture or other local image properties. Sometimes these methods are called "model-free", although the computer program used for tracking can be said to implicitly define a model.

Category 2: Methods using a 2D appearance model of a human being. The dimensionality "2D" relates to the fact that the model represents the human's appearance in the two-dimensional *image* as opposed to the 3D *space* it is moving in.

Category 3: Methods using an articulated 3D model of a human being, that is, a model of a person in 3D space.

Another aspect which can be modelled is the way in which the appearance of an object or its representation (e.g. model parameters) vary over time. An example of a temporal appearance model of a person has been given by Johnson 1998. The underlying assumption is that the appearance of a moving object varies in a regular manner over time and can therefore be modelled and predicted from frame to frame.

The more complex the model used for people detection and tracking, the better the system can handle the particular situations for which it is trained. However, with the level of complexity in the model the complexity of the computational problem also increases. The use of a 3D model means that the internal representation of the person cannot be taken from the image because an image only gives two-dimensional measurements. One approach to solve this problem involves the use of more than one camera, e.g. a stereo camera system, which in effect approximates 3D measurements of the person. Other approaches use iterative methods with projection from a 3D representation of the person onto the 2D image space. This makes the problem highly nonlinear and thereby often incompletely solvable and computationally expensive. It should be possible to speed up part of this process by using dedicated video hardware (e.g. existing 3D accelerated video graphics cards) for the projection 3D → 2D. However, the nonlinear and complex optimisation problem remains. Currently no such accelerated tracker exists.

The Reading People Tracker developed in this work is in Category 2.

2.2 Related Work

2.2.1 Overview and Classification

Realtime automated visual surveillance has been a popular area for scientific and industrial research since it was pioneered by O'Rourke and Badler (1980) and Hogg (1983). People tracking naturally plays a key role in any visual surveillance system, and a number of tracking algorithms for different applications have been presented (Baumberg, 1995; Brémond and Thonnat, 1997; Cai et al., 1995; Gavrila and Davis, 1996; Haritaoglu et al., 2000; Johnson, 1998; Khan et al., 2001; Lipton et al., 1998; Sidenbladh et al., 2000; Wren et al., 1995).

Using the classification scheme defined in Section 2.1.3 above, the tracking algorithms can again be classified into the three main categories of increasing model complexity:

Category 1: Methods using region- or blob-based tracking: Brémond and Thonnat (1997); Cai et al. (1995); Khan et al. (2001); Lipton et al. (1998); Wren et al. (1995).

Category 2: Methods using a 2D appearance model of a human being: Baumberg (1995); Haritaoglu et al. (2000); Johnson (1998).

Category 3: Methods using an articulated 3D model of a human being: Gavrila and Davis (1996); Sidenbladh et al. (2000).

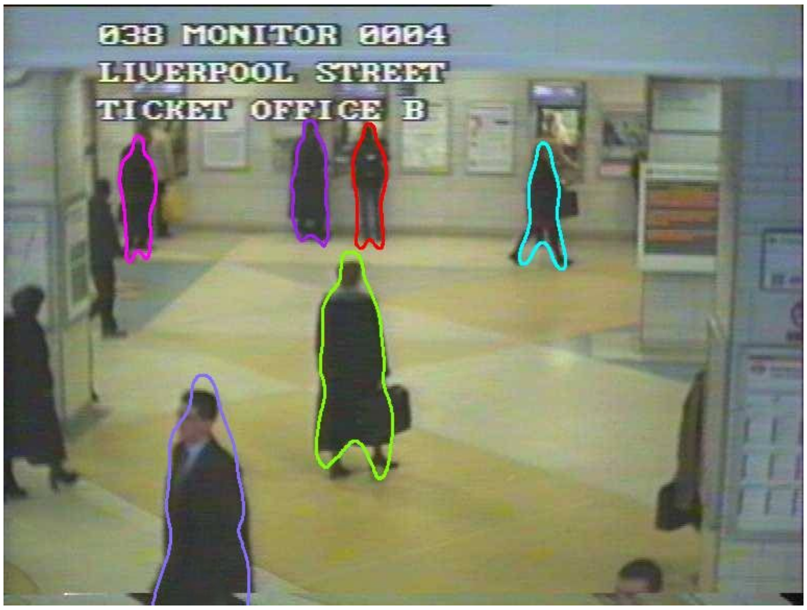


Figure 2.3: People tracked by the *Leeds People Tracker*

In the following, four examples of people trackers drawn from Categories 2 and 3 will be examined in more detail.

2.2.2 Baumberg's *Leeds People Tracker*

Adam Baumberg developed a people tracking system for his PhD at the University of Leeds under the supervision of Professor David Hogg (Baumberg, 1995). The tracking system, called the *Leeds People Tracker*, uses a 2D model of the shape of the outline of a person in the image. Therefore, it is in Category 2.

Overview of the Method

The people tracking algorithm is based on an *active shape model* of the contour of a walking pedestrian. The model is automatically generated in a training stage using a set of video images containing walking pedestrians (Baumberg and Hogg, 1995). This model generation involves the analysis of a large number of pedestrian outlines from training data using *Principal Component Analysis*, *PCA* (Gershenfeld, 1999, sect. 11.14). Figure 2.3 shows an example of tracked pedestrian outlines.

Detection

People detection is done in multiple stages, from low-level to high-level, as shown in the diagram in Figure 2.4.

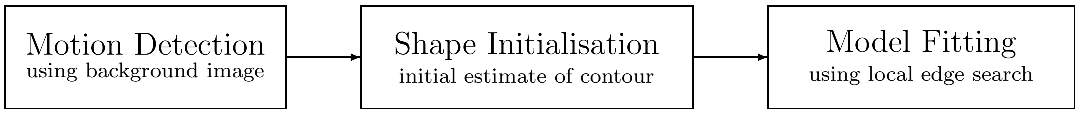


Figure 2.4: Detection hierarchy within the *Leeds People Tracker*

Step 1: Motion Detection

The motion detector used in the *Leeds People Tracker* is the *Reading Motion Detector* which was developed within the *VIEWS* project at the University of Reading[1]. It uses a simple image as a background model. The motion image is created by subtracting this background image pixelwise from a given video image and thresholding the result. A pixel in the background image is updated from the current video image if its value has been increasing for a number of consecutive frames. The rationale is that outdoor lighting changes usually create monotonically increasing or decreasing intensity values. Therefore, alternating changes are ignored. Areas in the image where people are currently tracked are masked out from motion detection, but not from background updating. The intention is that motion detection only detects new objects while old objects continue to be tracked. Depending on the available CPU time, the motion image can be filtered, e.g. using a spatial median filter.

Step 2: Shape Initialisation

For each new moving region coming from the motion detector that fits certain criteria, for example minimum and maximum size, an instance of the shape model is generated. It is represented by the control points for the *B-spline* which models the person's outline. This initial shape estimate (represented by model parameters) is usually obtained by using the trained mean pedestrian shape, scaled to the appropriate size. If camera calibration is available then it is possible to detect those regions which are too small to be a person. These are treated specially: Two hypotheses are then tested, either the detected region is the upper part of the person, or it is the lower part.

Step 3: Model Fitting

In the third and last stage of the people detection process, the current position and shape estimates are adjusted according to measurements in the image. The algorithm uses a local edge detection routine which operates in the pixelwise difference image

---

[1] The developers were Anthony Worrall (The University of Reading) and John Hyde (The Marconi Co Ltd). The algorithm was developed between 1990 and 1992 (Worrall, 2002).

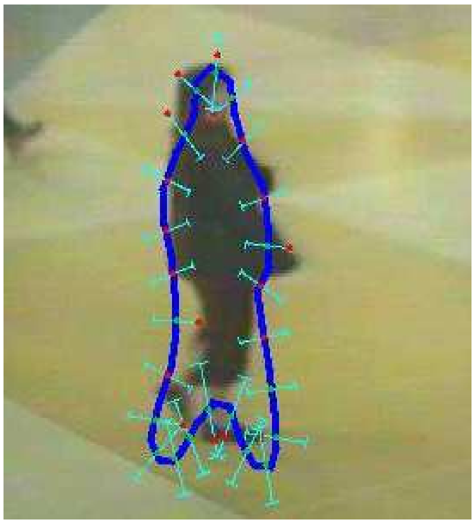


Figure 2.5: Edge search for shape fitting

between the background image and the current video image[2]. At a varying number of points around the current shape estimate measurements are made in order to find the contour of the person in the image. The search is carried out along lines which are aligned with the *Mahalanobis optimal search direction* (Baumberg, 1996). Figure 2.5 illustrates this process showing the original camera image, in this case with a dark person against a light background.

During this shape fitting process, the system allows for a certain percentage of the control points not to be matched in the current image. This ensures tracking when parts of the person are occluded, and the corresponding control points cannot be matched to edges in the image.

If the PCA model can be fit with a minimum fitness measure, it is decided the detected moving object in the image is a person. For each detected person in the image, the parameters for the PCA model, as well as the position of the person in the image are stored, and used in the tracking part of the code. Moving objects to which no shape can be fitted are discarded.

Tracking

To track people in subsequent images, the Leeds People Tracker uses a second order motion model, using a Kalman filter to model the speed and acceleration of a tracked person and to predict the position in the current frame. The initial positional estimate, together with the current shape parameters, are used as a starting point for determining the position and outline shape in the current frame. Repeated measurements made along the *Mahalanobis optimal search direction* (Baumberg, 1996) at the control points of the *B-spline* are used to find the new position and the outline of the person in the current image.

---

[2] In the implementation of the *Leeds People Tracker*, the difference image is not calculated explicitly, but this is the equivalent operation.

Advantages and Disadvantages of the Approach

The *Leeds People Tracker* has the following strengths:

Robustness: With an adaptive shape model and occlusion reasoning, a good level of detection and tracking robustness is achieved, as long people are reasonably separated from each other in the image.

Speed: Baumberg reports that the speed achieved by the tracker varies between 14.75 and 33 frames per second (fps) on an `SGI` workstation with a 64-bit superpipelined RISC CPU (MIPS R4000 clocked at 100 MHz) when using greyscale images at full PAL resolution (768 × 576 pixels), acquired from an external connector (Baumberg, 1995, 2002).

For most surveillance applications, this speed is sufficient for realtime operation, although it is unclear what the speed would be on more economical hardware using off-the-shelf PCs.

This people tracker and its implementation have been studied and tested in detail for this thesis. During this work the following disadvantages were discovered:

Limited Generality: The active shape model used in the *Leeds People Tracker* only models people's shapes if they are walking, and if a sufficiently large part of the body outline is visible. This means that the tracker cannot be used to detect or track sitting people, and it can have difficulties when tracking groups of people.

Initialisation of Tracks: When a new moving region is detected by the motion detector, generally one initial shape estimate is made[3]. This can create difficulties when two or more people who enter the scene are close in the image. One initial shape estimate will made in the centre of their common bounding box. This method can fail to detect any of them, especially if the application dictates a low frame rate which allows for significant object motion between individual frames.

---

[3] The only exception is the case discussed above, when the region is too small to contain a person. Then two initialisations are made, assuming the detected region is the upper or lower part of a person. At most one of the two initialisations is accepted as valid.

Occlusion and Low Contrast: The edge detector used in the shape fitting process relies on finding high contrast edges along the search direction. This fails when the person is moving in front of a similar-coloured background, resulting in non-detection (losing track) if a large percentage of edge points is not detected.

Re-gaining Lost Tracks: When a track is lost (e.g. due to occlusion or a person sitting down) and re-gained at a later stage, there is no mechanism to try to recognise the track as a re-acquisition of a previous track—neither from position nor from appearance.

Low Maintainability: The implementation of the *Leeds People Tracker* (approx. 50,000 lines of `C++` code) has a low level of maintainability. These problems are discussed in Chapter 5. They include dependence on specific hardware (an expensive `SGI` workstation), heavy use of global functions (which counters scalability) and lack of sufficient documentation.

Large Changes in Shape: The frame rate at which the *Leeds People Tracker* is run depends on the application and the number of objects in the image. If the frame rate is low (e.g. 5 fps as in the `ADVISOR` system), changes in human posture between consecutive images can be relatively large. This means that using the shape from the previous frame as an initial guess for the current shape can be inefficient or "misleading", resulting in errors detecting the person's current shape.

Evaluation

The Leeds People tracker uses a relatively complex system to initiate and accomplish tracking of a single person, allowing for partial occlusion. This works well if the image quality is good (for motion and edge detection) and if people are mainly isolated. However, tracking often fails if people overlap (e.g. go past each other) in the image, or if their outline shape is in not among those modelled by the active shape model. Once a track is lost for even one frame, there is no way of recognising the person if the track is re-acquired later.

2.2.3 Haritaoglu's W^4 System

The W^4 tracking and surveillance system is presented in Haritaoglu et al. (2000). It analyses *what* people are doing, *where* and *when* they do it and *who* is doing it. The main idea is to interpret and track silhouettes of foreground blobs using a feature-based approach. This makes this tracker belong to Category 2.

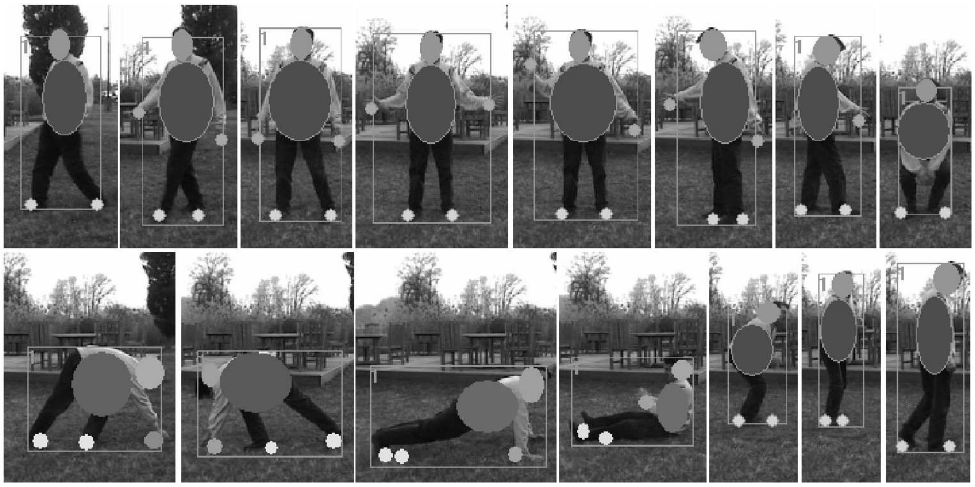


Figure 2.6: The W^4 system: Detected body parts for a number of body postures (image taken from Haritaoglu et al., 2000).

Overview of the Method

The tracking system has a complex structure, involving many different techniques. The following stages are used during tracking:

Stage 1: Detection

Good motion detection is crucial for this blob-based tracking system. Its background model uses a bimodal Gaussian model for each pixel. For each frame, pixels are classified as foreground or background using 2 thresholds, one for detecting "unusually high" inter-frame change and one for measuring the difference with the learned background distribution. The resulting foreground blobs are filtered using "region-based noise cleaning" and morphological filtering (a binary connected-component analysis).

Stage 2: Shape Analysis

Each of the connected blobs is classified into being either a *single person*, *people in a group* or *"other"*. This classification is based on the projection histograms of the blob, along and normal to the principal axis of the blob. The classification uses similarity measures comparing the histograms to *trained histograms* for single people and groups.

For *single people*, the posture is analysed using additional methods to the histograms mentioned above. At the initiation of the track a recursive convex/concave vertex algorithm (a *Graham Scan*, see Graham, 1972), is used, and in following frames a corner detection algorithm. If the person is in an upright position, symmetry and periodicity analyses are used to detect whether the person *carries an object*. If they do not, or if the person is not in an upright position, an algorithm to detect *body parts* is applied. This determines the position of the head, hands, feet and the torso of the person as well as 10 more detailed feature positions if they can be detected from the person's silhouette. A detected person is classified as standing, sitting, crawling/bending or lying down, and this classification is used to access a knowledge database about the positions of body parts of the person. Examples of detected body parts can be seen in Figure 2.6. The actual detection is based on the output from the convex/concave vertex or corner detection algorithm. An appearance model is built for each tracked person in order to recognise the person if their track is lost and re-acquired at a later stage.

For each *group of people*, the position of each person's head within the group is detected if it is exposed (that is, if the head outline is recognisable in the silhouette). In a second stage, the foreground region containing the group is segmented into subregions representing individuals using the local geometry of the silhouette. If people join a group, information about their appearance and texture is retained so as to recognise them once they leave the group.

Objects other than people are tracked, but not further classified.

Stage 3: Tracking

People are tracked using a second order motion model for their position (median coordinate) and an additional silhouette correlation matching (using the previous silhouette) using a 5×3 set of positional displacements.

Body parts, if detected, are tracked using temporal texture template matching and a second order motion model for each body part. Correlation techniques are applied to predict whether body parts will be occluded in the next frame.

Advantages and Disadvantages with this Approach

The people tracker used in W^4 has the following strengths:

High Versatility: The way the system models the appearance of a human being in the image is very general. Haritaoglu et al. (2000) show experiments with people who assume a number of very different postures. The tracker copes with these difficulties, correctly labelling head, feet and hand positions.

Integration of Behaviour Analysis: The system is integrated with behaviour analysis routines to detect simple interactions between people.

Speed: W^4 runs at a high frame rate of 20–30 Hz on a dual 400 MHz `Intel Pentium II` system, although at a relatively low image resolution of only 320 × 240 pixels.

Possible problems with this approach:

Reliance on Good Motion Detection: Since the tracking is based an accurate measurement of moving blobs, it needs a very good and robust motion detector. Detection, and therefore tracking fails when there are large shadows, much noise or large illumination changes. In the latter case the background model is recalculated to adapt to the new lighting situation (Haritaoglu et al., 2000).

Width of Field of View: The authors recommend at least a size of 75 × 50 pixels for body part analysis, multiple person tracking and carried object detection. At an image resolution of 320 × 240 pixels this limits the camera placement and viewing angle.

Reliance on High Frame Rate: Since tracking runs at near frame rate (20–30 Hz according to the authors) the system assumes that image changes between consecutive frames are small. This helps tracking. There is no information about tracking performance at lower frame rates.

Evaluation

Despite its complex structure, the system tracks people quite efficiently. The reported frame rate is 20–30 Hz at 320 × 240 pixels resolution when using grey-level video on a dual 400 MHz `Intel Pentium II` system. However, time-critical parts of the tracker, written in `C++` and Assembler code, are heavily optimised, using `Intel`'s *Streaming SIMD Extensions* (*SSE*) processor extension. If the output from motion detection is sufficiently good, the tracker can detect people, identify body parts of an isolated person and even find out whether they are carrying an object. It incorporates a good handling for groups of people, detecting silhouettes of exposed heads within each group. A database of measurements for each tracked person, created once they are standing or walking isolated from others helps to identify people if a track is lost and recovered at a later time.

This tracker does the maximum possible to track people without using an exact, parameterised 2D or 3D model of their appearance, and this works well.

2.2.4 Gavrila's 3-D Model-based People Tracker

Dariu Gavrila developed a 3-D model of a person and applied it to people tracking for his PhD under the supervision of Professor Larry Davis at the University of

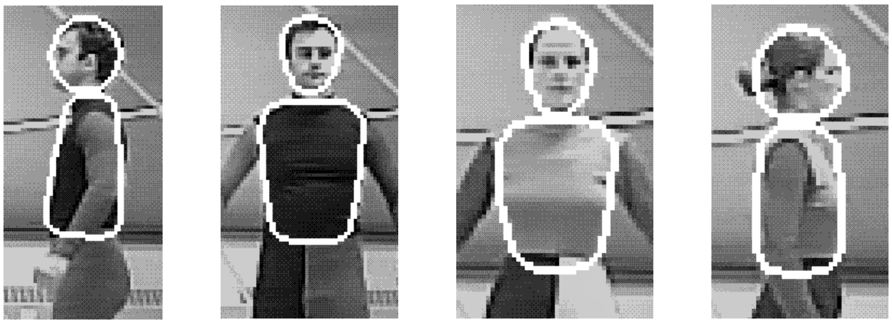


Figure 2.7: Gavrila's 3D tracker: Views of recovered torso and head (images taken from Gavrila and Davis, 1996, used by permission)

Maryland (Gavrila, 1996; Gavrila and Davis, 1996). The tracker uses a 22 DOF model of a person composed of *tapered superquadrics*, for example cylinders and ellipsoids. The shape is described by the joint angles between the superquadrics. Measurements are taken from 2 calibrated cameras with orthogonal viewing directions (for example, one looking at the person from the side, one from the front). In each camera view, segmentation of a 2D shape is done using the output from an edge detector. In the experiments people wore coloured, tight suits to ease edge detection (see Figure 2.7). As it uses a 3D model, Gavrila's tracker is in Category 3.

Overview of the Method

For initialisation of the track, a bootstrapping algorithm calculates the major axis (in 3D) of a moving blob observed in two camera views. This major axis is used to generate an initial estimate of the 3D appearance of the person.

In a second stage, and also during tracking, an edge-detection algorithm is used to create an edge-map of the image. Known edges (from the modelled background image) are removed, thus discarding most edges not related to the human shape. The actual model parameters (joint angles) are determined using an iterated search with projection to each 2D camera view, in a 3-stage process (find torso and head, find limbs, find arms and legs). To guide the search process, a *Chamfer* distance measure (Barrow et al., 1977) with a pre-processed distance lookup table is used.

Evaluation

Gavrila's tracker performs well, giving good tracking results. However, its specialisation makes it unsuitable for use in a surveillance application. The tracker relies heavily on the use of stereo sensor input (two cameras which even have to have orthogonal views). This renders the method unusable within a large-area surveillance application. Also, the edge detection used for fitting the person model relies

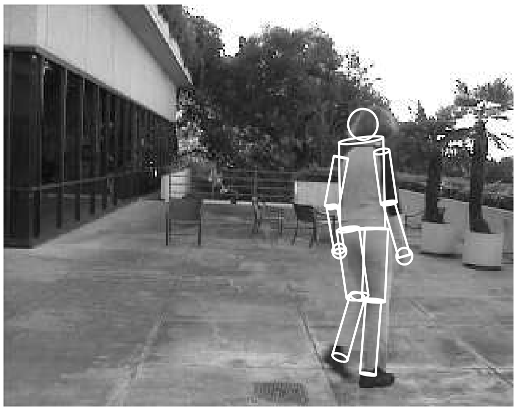


Figure 2.8: Sidenbladh's 3D people tracker: Superimposed model on tracked person (image taken from Sidenbladh et al., 2000, used by permission)

on reliably detecting edges around body parts. To ensure good edge detection, the participants wore special coloured suits during the experiments. Occlusion, apart from modelled occlusion (e.g., self-occlusion), is not handled by the tracker.

The idea to use a *Chamfer* distance measure seems interesting, and it reportedly works better for this application than correlation and average contrast because it can provide more precise off-peak responses for large search areas. It seems well-suited to match model edges to an edge image, and using a pre-computed lookup table makes it fast, too.

The execution speed of the tracker is not given. It seems that the tracker was run offline on digitised sequences, not on live input.

2.2.5 Sidenbladh's 3D People Tracker

In her PhD work, Hedvig Sidenbladh developed a method for the modelling and tracking of human motion using a complex articulated 3D model of a person from a sequence of video images from a single camera (Sidenbladh, 2001; Sidenbladh et al., 2000). The work was carried out at the Kungliga Tekniska Högskolan (KTH) in Stockholm and at Brown University (Providence, USA), under the supervision of Jan-Olof Eklundh, KTH and Michael Black, Brown University. It is in Category 3.

Overview of the Method

Tracking is divided in two parts: Firstly, a probabilistic model is estimated using data of typical activities obtained from a large set of 3D human motion data. In a second step, this probabilistic model is used as a prior distribution for Bayesian propagation using particle filters.

The human body is represented as a set of articulated cylinders with 25 DOF, see Figure 2.8. For repetitive human motion such as walking, motion sequences are decomposed into a sequence of temporal models known as “motion cycles”. To build these temporal models, training data is automatically segmented using the mean and the principal components of the individual cycles in the reference domain as a probabilistic model. In order to deal with missing information in the motion time-series and to enforce a smooth transition between different cycles, an iterative method for data imputation and functional Principal Component Analysis (PCA) is used, based on periodic regression splines.

The trained temporal model provides a prior probability distribution over human movements which is used in a Bayesian framework for tracking. A generative model of image appearance is constructed from warped image patches wrapped around the cylindrical elements of the body model. It is used to determine the likelihood of observing modelled image data. The high dimensionality and non-linearity of the articulated human body model and the ambiguities in matching the generative model to the image result in a posterior distribution that cannot be represented in closed form. Therefore, the posterior is represented using a discrete set of samples and is propagated over time using particle filtering. The learned temporal prior helps to constrain the sampled posterior to regions of the parameter space with a high probability of corresponding to human movement.

Evaluation

Sidenbladh’s tracker has been validated by experiments with 175 frames at an image resolution of 320×240 using 30 frames per second of video (Sidenbladh et al., 2000). The experiments show that the algorithm tracks human motion reasonably well, see Figure 2.8. The trained temporal model for a walking cycle is used to constrain the search for a parameter set in the high-dimensional search space, yet the number of function evaluations (parameter estimation in 25-dimensional space plus projection 3D → 2D) for each frame, given as 10,000 to 15,000, is still very high.

The high complexity of the model (for example, using warped images for the cylinder surfaces) makes the algorithm very computationally expensive. One problem arises when clothes are not very tight, so the shape of the sleeve on the underarm might look quite different from the predicted shape. One other, major problem is the immense computational cost involved to match the model against the sensor data. The reported execution time for each frame on a `Sun Ultra 1` workstation is 5 minutes, sampling around 10,000 states. The tracker is made with applications within the movie industry in mind. Even if some optimisations were done it does not seem suitable for surveillance applications.

2.2.6 Conclusions

The more detailed the models for people detection and tracking, the better the system can handle the particular situations for which it is trained. However, systems in Category 3 with complex 3D models, e.g. Sidenbladh's 3D People Tracker, are currently too slow to use in realtime systems. They might also require a special camera/scene setup, like Gavrila's 3-D Model-based People Tracker. This is rarely available in a CCTV environment. This is why most of the methods used for visual surveillance fall into Categories 1 or 2.

There are different approaches within Category 2 methods which use a model of the 2D appearance of a human in the image. Baumberg's *Leeds People Tracker* uses a model of the outline shape of a person in the image. This works well as long as the person is clearly visible and no more than slightly occluded. Haritaoglu et al.'s W^4 system works mainly in a low-resolution motion image. It requires a very robust and precise motion detector. Given that, it works very well and correctly tracks and labels people and their body parts. Both of these 2D methods are fast enough for use in a realtime system.

2.3 Building an Integrated People Tracking Application

Building a commercial visual surveillance system which includes people tracking and behaviour analysis is a complex task. This section discusses the general problems and introduces the solutions which were incorporated into the `ADVISOR` system.

2.3.1 The `ADVISOR` Project and its Objectives

Three academic and three industrial partners have worked together in the European Union's research project `ADVISOR`[4] to build an integrated realtime surveillance system for use in underground stations. The time span for the completion of the project was three years, January 2000–December 2002. The main requirements of the `ADVISOR` system are as follows:

1. Track people in camera images from multiple cameras
2. Estimate crowd density and motion
3. Analyse people and crowd behaviours based on these estimates

---

[4] Annotated Digital Video for Intelligent Surveillance and Optimised Retrieval, EU project code IST-1999-11287, project web site `http://www-sop.inria.fr/orion/ADVISOR/`.

4. Generate alarms based on detected dangerous or criminal behaviours and display these to a human operator within a short time (limited delay)
5. Store all digital video images and event annotations in a database and allow for searches within this database through a graphical user interface
6. The system has to be a fully integrated unit with a single interface to an underground station's camera system.

Below is a list of the six partners with their areas of expertise and main contributions to the overall system:

1. The Computational Vision Group at The University of Reading has competence on the visual surveillance of people and vehicles (Remagnino et al., 1997). This group has developed a People Tracker for `ADVISOR` which is presented in this thesis.
2. Sergio Velastin's group at King's College London (now at Kingston University) has worked extensively on crowd monitoring and analysis (Yin et al., 1994). With this experience, the group has implemented Crowd Density and Motion Estimation functionalities.
3. The ORION research group at INRIA Sophia Antipolis has expertise on behaviour analysis and event detection in surveillance applications (Chleq et al., 1998). ORION provided the interpretation part in form of a Behaviour Analysis module.
4. The Belgian company Vigitec are suppliers and maintainers of visual surveillance systems. They have worked together with the ORION group in the past and now took over Human-Computer Interface (HCI) design and implementation, as well as establishing contacts with the end users (station operators).
5. The French computer company Bull has expertise in databases and storage solutions and hence provided a Video and Annotation Database for `ADVISOR`.
6. The UK based company Thales Research (formerly Racal Research) managed the project and was responsible for the important task of system integration as well as the Video Capturing facilities of the `ADVISOR` system.

2.3.2 System Overview

The system requirements, i.e. the functionality of the `ADVISOR` system was agreed before the project had started. It was decided that the system was to be divided into

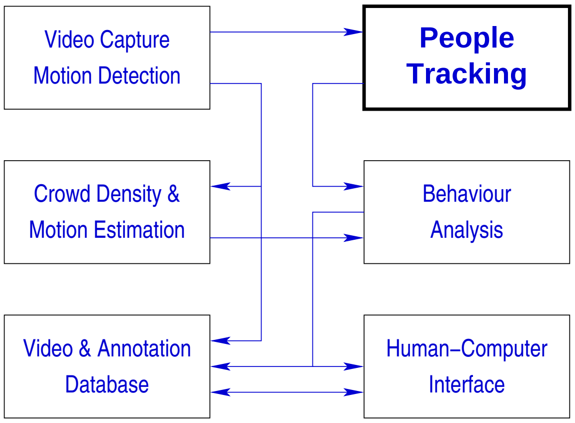


Figure 2.9: Overview of the six subsystem of the `ADVISOR` system

six subsystems, one for each major functionality. The individual subsystems were to be connected via 100BaseT Ethernet (up to 100 Mbit/s) so that they can exchange data. Figure 2.9 shows an overview of these subsystems and the data flow between them.

Video Capture and Motion Detection: Built by Thales and INRIA, this subsystem is connected to a video feed from the camera system in the underground station. It captures the video and sends it to all other machines using IP Multicast. During the project it was also decided that motion detection also be done on this computer so that it can be done directly on live images. INRIA have implemented their motion detector here for this purpose.

People Tracking: The People Tracking subsystem uses the video and motion data to locate and track all people in the camera images.

Crowd Density and Motion Estimation: The Crowd Analysis subsystem uses motion estimation hardware to analyse the movement and density of crowds. Example applications are to detect overcrowding, blockages and reverse movement in one-way corridors.

Behaviour Analysis: Based on both short-term and long-term analysis of the video annotations generated by People Tracking and Crowd Analysis, events are recognised and alarms generated by the Behaviour Analysis module. Events to be recognised are fighting, blocking the way, overcrowding, jumping over barriers and vandalism against machines.

Human-Computer Interface (HCI): The HCI displays any event detected by Behaviour Analysis, together with the relevant video images, to the human operator.

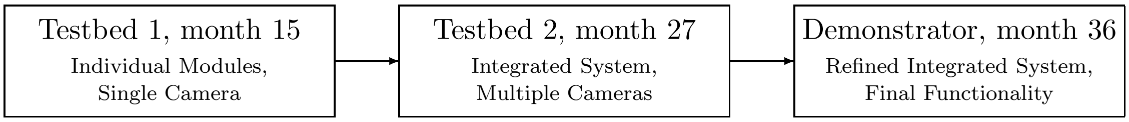


Figure 2.10: Incremental stages and requirements within the `ADVISOR` project

Video and Annotation Database: The Video and Annotation Database stores all video and annotations generated by the system. It allows to query the database based on a number of search criteria through the HCI.

Five of the six subsystems communicate with each other over the Ethernet using the *CORBA* suite of specifications for communication. The exception is the HCI. It is written in `Microsoft Visual Basic` which does not support *CORBA*. The connections with the HCI are done using some Microsoft-specific pipes. All other subsystems are written in `C` and `C++`. The system is specified to handle the input from 4 cameras simultaneously.

2.3.3 Development Plan

It was decided to implement the overall functionality incrementally. An incremental approach to system development is widely considered to be a good way to build a system (Moore and Backwith, 1999). One of the benefits is the possibility for better risk management through an early assessment of finished parts of the system. Specified parts of the system can be delivered and tested against *Use Cases* which define how a system's users interact with the system. These incremental Use Cases are part of a path to the final system and its Use Cases. For example, the software manufacturer ARTiSAN Software Tools has incorporated the concept of incremental development into their *Real-Time Perspective* development model for realtime systems (Moore and Backwith, 1999). The authors point out the advantages and practicability of this approach and provide a set of development techniques for its application to realtime systems.

Figure 2.10 shows the initial plan for the most important `ADVISOR` incremental project deliverables. After 15 months, the six individual modules shown in Figure 2.9 were to be tested "offline" in *Testbed 1*. "Offline" means that instead of using live video and annotation input, all input and output data is read from and written to a local hard disk. This way, the functionality of the individual modules for people tracking, crowd analysis and behaviour analysis can to be tested without the need to integrate the system at this stage. The requirements are for a single camera input. After 27 months, these individual modules had to be fully integrated as subsystems of the `ADVISOR` system. Realtime tracking and analysis was to be demonstrated in *Testbed 2*, using simulated live video input from multiple video recorders. Based on the validation of the demonstrated systems final adjustments and improvements were to be made and presented in the final *Demonstrator* in month 36. This Demonstrator was to be presented in an underground station in Barcelona, running on live video input from 4 cameras.

2.3.4 Data Formats for Communication

The data packets to be exchanged between the six subsystems shown in Figure 2.9 are of two types: images (video and motion images) and textual data, for example tracking results and event annotations.

The Need for Image Compression

The video images are transmitted in digital form over Ethernet. If this were to be done without image compression, this would require a high data rate. For example, transmitting 3 uncompressed video streams at 5 frames/second (fps) and PAL resolution (768 × 576 pixels) in 24-bit colour requires a sustained data transmission rate of

$$3 \cdot \frac{5}{\mathrm{s}} \cdot 768 \cdot 576\,\mathrm{pixel} \cdot \frac{3\,\mathrm{Byte}}{\mathrm{pixel}} = 19{,}440\,\frac{\mathrm{KByte}}{\mathrm{s}} = 151.875\,\frac{\mathrm{Mbit}}{\mathrm{s}}. \qquad (2.1)$$

The standard for Ethernet nowadays is 100BaseT, which supports data rates up to $100\,\frac{\mathrm{Mbit}}{\mathrm{s}}$. Experts suggest that in any realtime system, the required load of a link should not be above 40–50 % of the maximum link speed (see e.g. EventHelix.com Inc, 2000–2002) which gives an availability of $50\,\frac{\mathrm{Mbit}}{\mathrm{s}}$ for video plus annotations. This is far too little, and therefore in our system the video images are compressed before sending them across the Ethernet network. This compression can take place in software or hardware (e.g. in an add-on module for the frame grabber card which digitises the video).

The image compression algorithm used in our system is *JPEG* (see e.g. Lane, 1991–1995) which reduces the data rate to an acceptable amount. JPEG uses a discrete cosine transform (*DCT*) to retain only “essential information” in 8 × 8 blocks of pixels. “Essential information” is defined by a quality factor which is related to the visual accuracy of the image, and determines the compression ratio—from about 2 : 1 up to 100 : 1. JPEG compression is called a “lossy” compression method because it discards information which cannot be restored at a later stage. One particular problem is that JPEG compression techniques usually sub-sample and quantise colour information (the 2 “chrominance channels”, denoted C_B and C_R, ibid.). The implications of this JPEG compression on image processing routines will be discussed in Chapter 3.

Results and Annotations

It was decided to use *XML* for all textual data exchange between the subsystems. *XML* is a simple, open standard by the Word Wide Web Consortium, W3C, (see `http://www.w3c.org/`), which lets two computers exchange any type of data in an easy way. One of the reasons for choosing XML was the possibility to standardise an interface through the use of an *XML Schema*. An *XML Schema* is a special XML document which defines how the XML data for a particular data channel will look. Once an XML Schema is agreed on, both the deliverer and the receiver of XML data can easily validate whether a given piece of XML data adheres to the agreed XML Schema. This avoids the dangers of missing data or misinterpretation of data. A range of tools exist to automatically validate an XML file against a given XML Schema.

2.3.5 Building a People Tracking Module for `ADVISOR`

The Computational Vision Group at the University of Reading has built a People Tracking subsystem for `ADVISOR`. This thesis presents the research and development work carried out on the subsystem. The new Reading People Tracker is based on the *Leeds People Tracker* introduced in Section 2.2.2. The *Leeds People Tracker* is robust and fast and provides a good basis for further development. The original source code of the tracker was given to the Computational Vision Group by the University of Leeds when Reading and Leeds were partners in a joint research project (Remagnino et al., 1997).

While its basic tracking algorithm is very good, the *Leeds People Tracker* lacked a number of features which were needed within `ADVISOR`. The main new requirements for the People Tracking subsystem were as follows:

- The `ADVISOR` system requires the People Tracker to operate in realtime on a standard off-the-shelf hardware PC to enable economical system integration.
- The People Tracker has to run multiple trackers in parallel, for video input from multiple cameras (original software: one tracker, one camera input).
- The People Tracker has to be fully integrated into the `ADVISOR` system, exchanging data in the agreed formats (XML, JPEG).
- When running as a subsystem of `ADVISOR`, there is no monitor and keyboard attached to the People Tracking PC. Therefore all graphical output and all required user input has to be disabled.
- The People Tracker has to either accept motion detection input from the external Motion Detection module or use its own Motion Detector.

- Good software documentation.

The following issues made it hard to meet the above requirements.

- The heavy use of global variables and functions in the *Leeds People Tracker* meant that multiple cameras could not be used.
- Except a few pages of operating manual, no other documentation was available.
- Although the code was written in `C++`, it made a limited use of object oriented features like encapsulation and inheritance.
- There were only few comments in the source code, making the code difficult to read.
- The names of some of the classes and methods were misleading.
- The code was written for an `SGI` platform running under `IRIX`, making use of hardware- and software-specific functions.

It was therefore decided to re-design the People Tracker and port it to a PC running `GNU/Linux` before adding the necessary new functionality for `ADVISOR`. PCs provide a good platform for economical system development and deployment, and using `GNU/Linux` as an operating system eases the transition from the `SGI/IRIX` system.

Chapter 3

Colour Image Filtering for Robust Image Processing

The People Tracker presented in this work uses a motion detector, and local edge search in a shape fitting process. This chapter presents experiments which show how image quality influences shape fitting, and how image filtering techniques can help to overcome these problems.

3.1 Introduction

The images which are used by the People Tracker travel a long way from the point where they are first taken by a camera. The transmission and digitisation of the images introduces noise which can influence image processing routines. In order to apply countermeasures, one has to determine the exact nature of the noise, and then assess whether and how it can be dealt with. This chapter examines the way noise is introduced into the images in our surveillance system. Experiments show how image filtering can minimise the influence of this noise on image processing algorithms. The results presented here have been published in Siebel and Maybank (2001a) and Siebel and Maybank (2001b).

3.1.1 The Representation of Colour in Computers

The colour of a picture element (pixel) can be represented within a computer in an number of ways. Generally, a particular colour is considered to be an element of a three-dimensional *colour space*, with each component taking values in the range 0..255. The components are represented by unsigned 8-bit integers. Travis (1991, chap. 3) defines 8 colour spaces. Among these the following are commonly used:

RGB: The *RGB colour space* is used by most colour computer displays and most colour cameras. A colour is represented by *red*, *green* and *blue* components, corresponding to the relative proportions of the display primaries required to produce it. Although easy to implement it is nonlinear with visual perception (Ford and Roberts, 1998).

HSV: The representation of colour in the *HSV colour space* (similar concepts: *HSI*, *HSL*) space is a polar coordinate model which is very intuitive for humans. It consists of splitting the colour into almost "visually orthogonal" *hue*, *saturation* and *value* (*intensity*, *lightness*) components.

If the *red*, *green* and *blue* directions in RGB space are made Euclidian standard basis vectors then the major diagonal of the unit cube, from black at $(0, 0, 0)$ to white at $(1, 1, 1)$ forms an achromatic axis or *greyscale*. If the cube is viewed along the diagonal from the black corner to the white corner then a hexagon is seen with hues radiating from the achromatic axis through the centre of the hexagon. The HSV colour space uses this concept to define a *hue* angle, *saturation* (distance from the achromatic axis) and a third parameter, *value*, which broadly corresponds to lightness (Lilley et al., 1993).

$\mathbf{YC_BC_R}$: The YC_BC_R *colour space* is one of a family of *luminance-chrominance*-based colour spaces (e.g. *YIQ*, *YUV*, *YCC*). These spaces split a colour into two main components: *luminance*, commonly denoted by the symbol Y, is the perceived brightness, whereas the *chrominance* is the "colouredness". The *chrominance* itself is decomposed into two sub-components which differ between the spaces. These spaces are widely used in compression applications, both digital (e.g. *JPEG*, *MPEG*) and analogue (e.g. video transmission using *composite* signals). These standards were mainly designed for television transmission with the intention to provide television signals compatible with black and white television sets. The *CIE*[1] has published a number of recommendations which define these standards, depending on the hardware (camera, monitor) used in an application.

The main colour space used within the Reading People Tracker is RGB because this is the common representation of colour images within a computer. However, there are also algorithms which make use of other colour spaces, and these have been used for the experiments described in this chapter.

---

[1] Commission Internationale de L'Éclairage

3.1.2 Image Generation and Transmission

Figure 3.1 shows the stages the individual video images go through in the `ADVISOR` system, from the moment they are taken by a camera to the moment they arrive as an uncompressed digital image in the memory accessed by the People Tracker.

Video cameras: The video cameras in the station generate analogue video signals using a CCD sensor. In digital cameras this signal is digitised before any further processing is done. At present, most cameras found in underground stations are analogue, and this is the case examined here.

Analogue cabling: The analogue video signal is transmitted down long analogue cabling. Nowadays the composite video signals are usually transferred using optical fibre cables, avoiding electrical interference at this point. However, some signal degradation remains because a number of signals (often 32 or 64) are multiplexed using frequency modulation for transmission through a single fibre channel.

Video matrix switcher: The signals from multiple cameras in the underground station are usually fed into a *video matrix switcher*. This switcher allows images from selected cameras to be displayed to monitors or sent to recording or processing equipment—in our case the `ADVISOR` system. Modern video matrix switchers often allow an image overlay with text, e.g. station identifiers, and a time display. Image overlays can also be done by a separate module, either before or after the switcher.

The switching and the addition of text overlays to analogue video signals invariably introduce some noise to the signal (and hence the image), although the noise might be negligible.

Video recording and playback: Video data from the switcher is often recorded using a video cassette recorder (*VCR*). Usually the VCR uses a `vhs` or a `s-vhs` format to store the video signal in an analogue form. Both formats subsample colour information and reduce the resolution of the original signal, significantly degrading quality of the video images. `s-vhs` keeps more of the original information and hence gives slightly better results.

Digitising the image: Video images are digitised using a *frame grabber*, usually a simple add-on card for a computer. It delivers an uncompressed digital representation of the image into the computer's memory, or to another add-on card or module.

JPEG compression: Image compression can take place in software or hardware (e.g. in an add-on module for the frame grabber itself). The compressed images are transmitted to image processing computers via 100BaseT Ethernet.

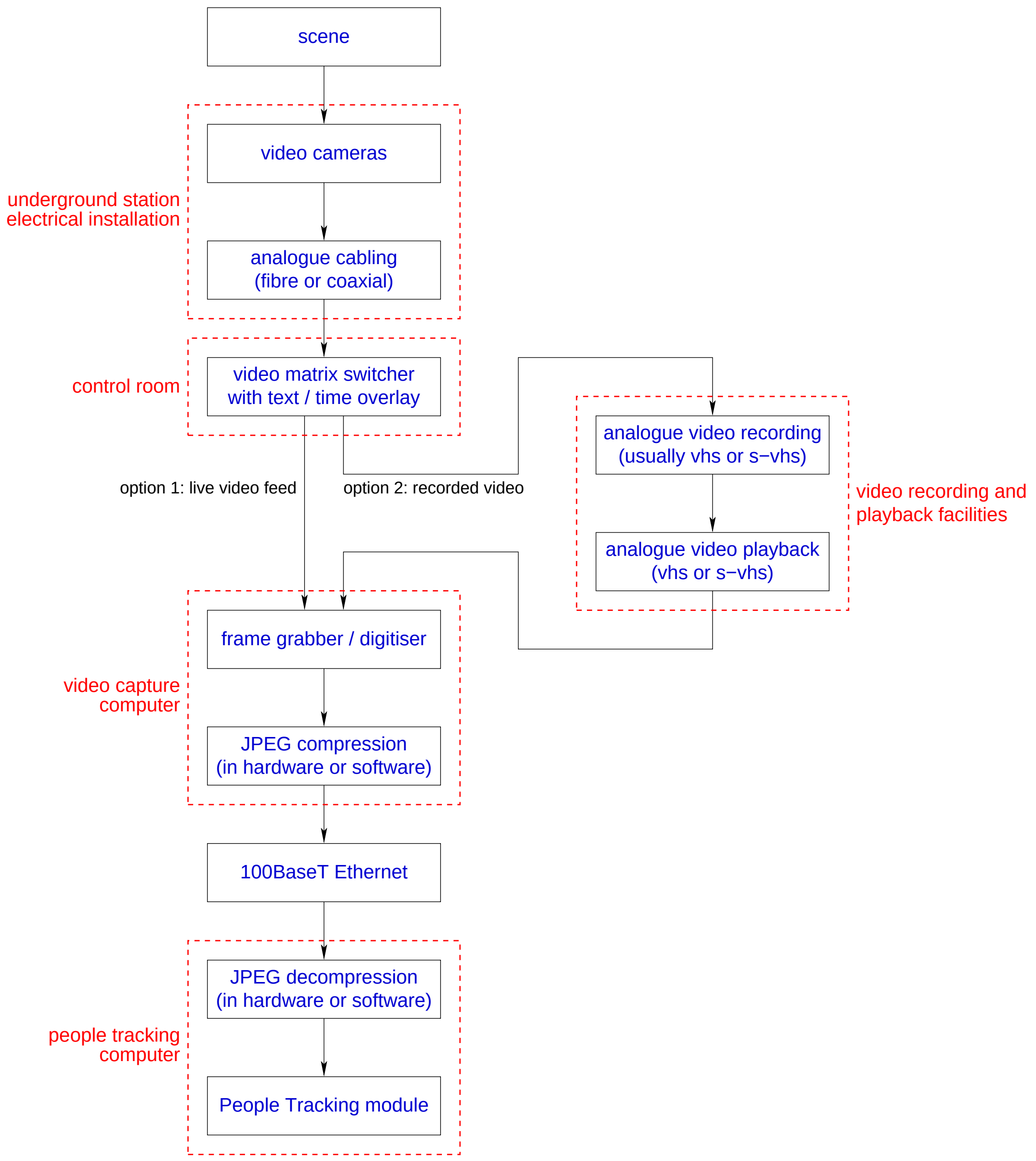

scene
video cameras
underground station
electrical installation
analogue cabling
(fibre or coaxial)
control room
video matrix switcher
with text / time overlay
analogue video recording
(usually vhs or s−vhs)
option 1: live video feed
option 2: recorded video
video recording and
playback facilities
analogue video playback
(vhs or s−vhs)
frame grabber / digitiser
video capture
computer
JPEG compression
(in hardware or software)
100BaseT Ethernet
JPEG decompression
(in hardware or software)
people tracking
computer
People Tracking module

Figure 3.1: The path of an image from the video camera to the tracker

The image compression algorithm used in our system is JPEG (see e.g. Lane, 1991–1995) which reduces the data rate by a factor between 2 : 1 and 100 : 1, but also introduces significant image degradation. One particular problem is that JPEG compression techniques usually sub-sample and quantise colour information (the 2 "chrominance channels", denoted C_B and C_R, ibid.). The effect of this will be more closely examined below.

Transmission over Ethernet: The digitised and compressed video images are sent from the video capture computer to the people tracking computer using 100BaseT Ethernet. The reduction of data size through image compression makes this possible. Thanks to the digital nature of the transmission, no further image noise is introduced at this stage.

JPEG decompression: In order to process the images, they are decompressed from JPEG to a raw format and made available to the People Tracking module. As with compression, image decompression can take place in software or hardware. No more noise is introduced at this stage.

3.1.3 Determining the Effects of Image Noise

It is easy to assess whether noise is introduced at each of the stages presented above. It is more difficult to assess the influence of noise on image processing algorithms. For example, JPEG image compression algorithms are designed to discard some colour information from the image because this is much less important to the human eye than brightness information. Depending on the exact implementation of an image processing algorithm, it might or might not rely on the colour information discarded by JPEG compression. Therefore, an image can look acceptable to a human eye while image processing algorithms are seriously affected because they rely on missing or badly interpolated data.

3.2 Image Noise and Image Filtering

3.2.1 Use of Images Within the People Tracker

Figure 3.2 shows how a video image is used by the People Tracker. The Motion Detector uses the video image to generate a number of other images. Some of these are strongly affected by image noise, others are less affected.

- The *Background Image* is created by filtering the image over time using an approximation to a median filter. "Outliers" in pixel values are removed. After the filtering the noise introduced by the cameras and cabling does not influence the overall Background Image much.

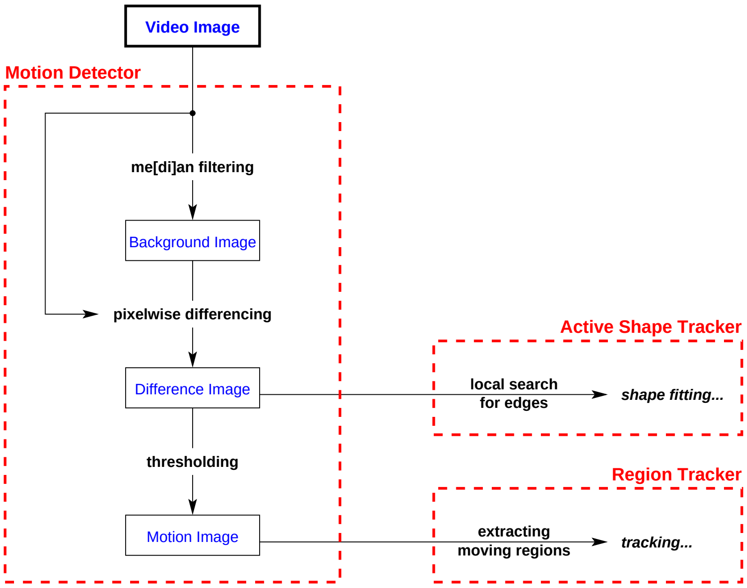

Video Image
Motion Detector
me[di]an filtering
Background Image
pixelwise differencing
Difference Image
thresholding
Motion Image
Active Shape Tracker
local search for edges
shape fitting...
Region Tracker
extracting moving regions
tracking...

Figure 3.2: The way a video image is used by the People Tracker

- The *Difference Image* is a monochromatic (*greyscale*) image. Its creation involves the pixelwise absolute difference between the Background Image and the Video Image (see Section 3.2.2 for differencing techniques). For this reason, any noise in the Video Image is also found in the Difference Image.

- The *Motion Image* is created by thresholding the Difference Image. Therefore the influence of the noise on the Motion Image is dependent on the noise amplitude.

The Difference Image is used by the Active Shape Tracker during the shape fitting process (see Section 2.2.2 for details). In particular, pairs of pixels on a *search line* around the contour are examined to see if there is an edge between them. Figure 3.3 illustrates this local search for edges: The value of a pixel a in the Difference Image (the right image) is compared to an "inner threshold". If the value is larger than the threshold the pixel is considered to be inside the person. Another pixel b's value is compared to an "outer threshold". If it is smaller, b is considered to be outside. This is done in an optimisation loop for a restricted number of point pairs (a, b). There is insufficient CPU time to evaluate all pixels along the search line.

The Difference Image is the most relevant image to the People Tracker. Therefore it is important to find methods to reduce the influence of image noise in this image. The way in which the Difference Image is used also clearly suggests a measure for the negative impacts of image noise: A "good" Difference Image is one in which the people are clear and salient (i.e. light) against a dark background, such that any

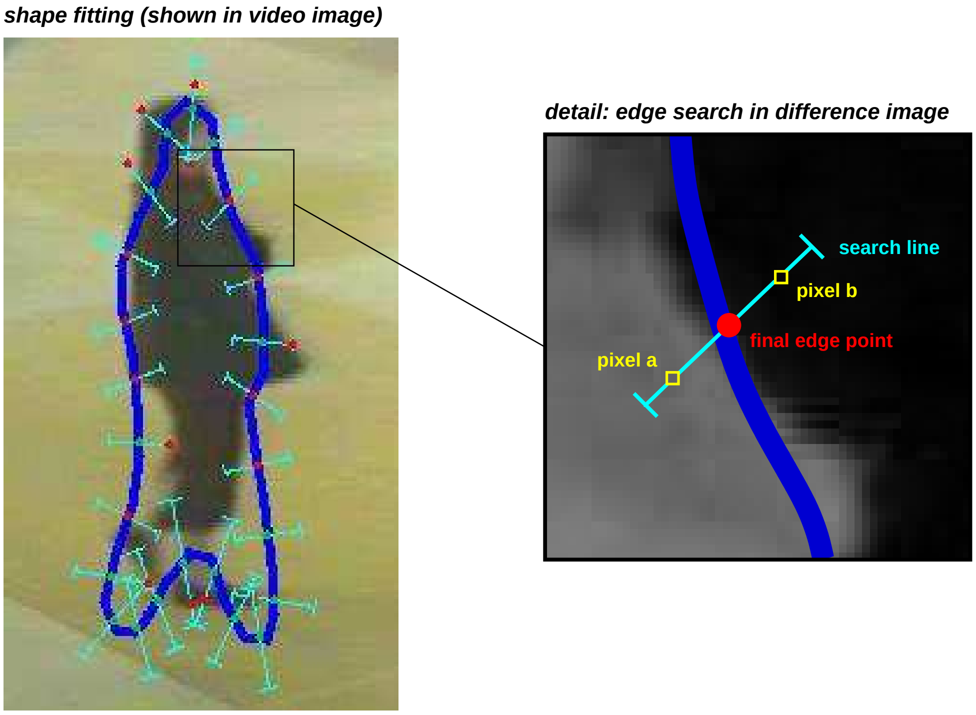

shape fitting (shown in video image)
detail: edge search in difference image
search line
pixel b
final edge point
pixel a

Figure 3.3: Local search for edges along a search line during shape fitting

search for an edge around the contour of a person gives a correct response.

3.2.2 Filtering Techniques: Creating the Difference Image

3.1. Definition (LOCAL IMAGE FILTER): *A local image filter* F *is a method that performs a transformation on an image* I_1 *in order to produce another image* I_2 *of the same dimensions,* $I_2 = F(I_1)$ *such that the value of a pixel at position* (x, y) *in the new image* I_2 *only depends on the pixel values in an* ε*-neighbourhood* $N_\varepsilon(x, y)$ *around pixel position* (x, y) *in the original image* I_1.

In applications the ε-neighbourhood $N_\varepsilon(x, y)$ is usually small compared to the whole image, e.g. a 3×3 pixel window which has (x, y) as its centre. Typical examples of local image filters are *smoothing* (*blurring*), e.g. using a Gaussian kernel, and changing the brightness of an image, e.g. γ-*correction*. See Forsyth and Ponce (2003) for these and other examples.

The operation which creates the Difference Image starts with two RGB colour images, the Video Image and the Background Image. Each pixel value $p = (R, G, B)$ in these two colour images, with red, green and blue components is an element of $\mathbb{R}^3$. The Difference Image is monochromatic, so its pixel values lie in $\mathbb{R}$. A local image filter used at this point would therefore have to map an RGB colour image

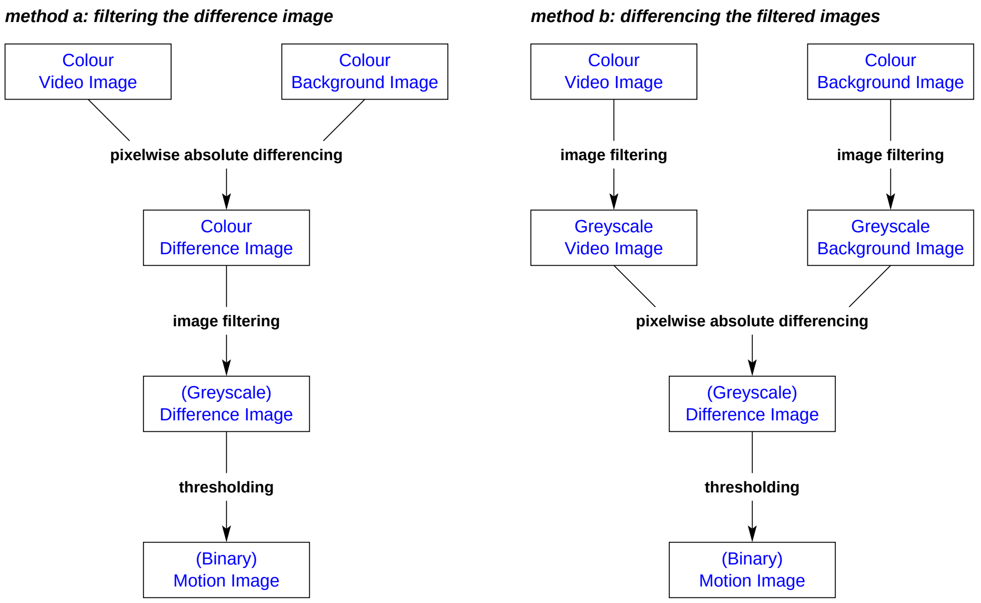

method a: filtering the difference image
Colour Video Image
Colour Background Image
pixelwise absolute differencing
Colour Difference Image
image filtering
(Greyscale) Difference Image
thresholding
(Binary) Motion Image
method b: differencing the filtered images
Colour Video Image
Colour Background Image
image filtering
image filtering
Greyscale Video Image
Greyscale Background Image
pixelwise absolute differencing
(Greyscale) Difference Image
thresholding
(Binary) Motion Image

Figure 3.4: Alternative methods to produce the Difference Image

onto a greyscale image. We will only consider local image filters where each pixel value $I_2(x, y)$ in the new (filtered) image only depends on the value of pixel $I_1(x, y)$ at the same position in the original image. The local image filter can therefore be defined by a *mapping function* $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ which maps pixel values in the original image to pixel values in the new image:

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}, \quad I_1(x, y) \mapsto I_2(x, y). \tag{3.1}$$

The mapping function f uniquely defines a corresponding local image filter F.

The Difference Image can generally be produced in two ways, as illustrated in Figure 3.4. *Method a* is to apply the local image filter F to the Colour Difference Image, obtained by differencing the Colour Video Image and the Colour Background Image, using absolute differencing in each component of (R, G, B). Given the colour pixel value (R_1, G_1, B_1) in image I_1 and the value (R_2, G_2, B_2) in image I_2 the absolute difference is defined componentwise:

$$\text{absdiff}((R_1, G_1, B_1), (R_2, G_2, B_2)) := (|R_1 - R_2|, |G_1 - G_2|, |B_1 - B_2|). \tag{3.2}$$

In *Method b* both the Colour Video Image and the Colour Background Image are filtered, giving two greyscale images. These are then differenced to generate the Difference Image. For the generation of the Difference Image, this can be equivalent to using a camera which supplies greyscale images. This depends on the image filter

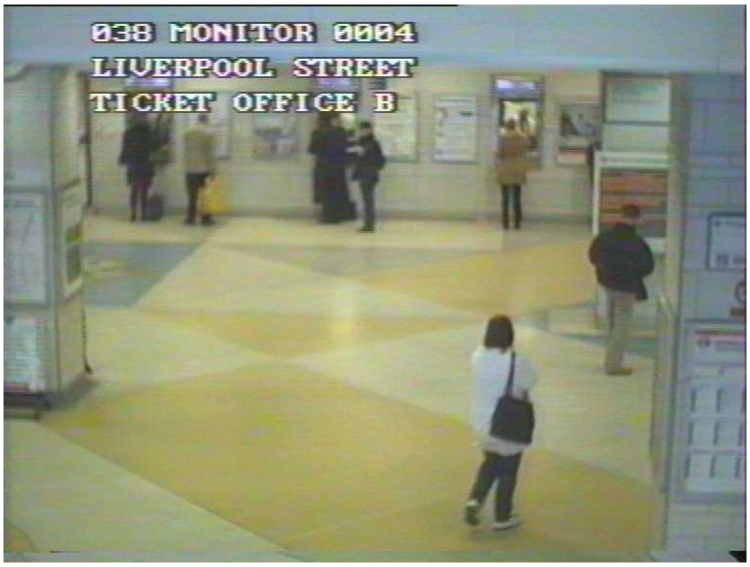


Figure 3.5: Video image used for colour filtering experiments

which is used. (Note, however, that other parts of the People Tracker still use the available colour information.)

The *Leeds People Tracker* uses *Method a*. Our People Tracker has been equipped with the option to use either method. Section 3.3 presents results from experiments with both methods and a number of local image filters, with the aim of finding an optimal combination of a local image filter and one of the two methods to minimise the effects of image noise.

3.3 Experiments and Results

3.3.1 The Test Data

The investigations involved a large number of colour filtering techniques and images. The image in Figure 3.5 was chosen as one of the test images. It is a typical image and has most of the difficulties which are important for these experiments. Below is some information on the image.

- Image resolution: PAL (768 × 576 pixels), RGB colour, taken by an interlaced camera at London Underground's Liverpool Street station.
- The video signal was transmitted to the control room in analogue form (composite video signal).

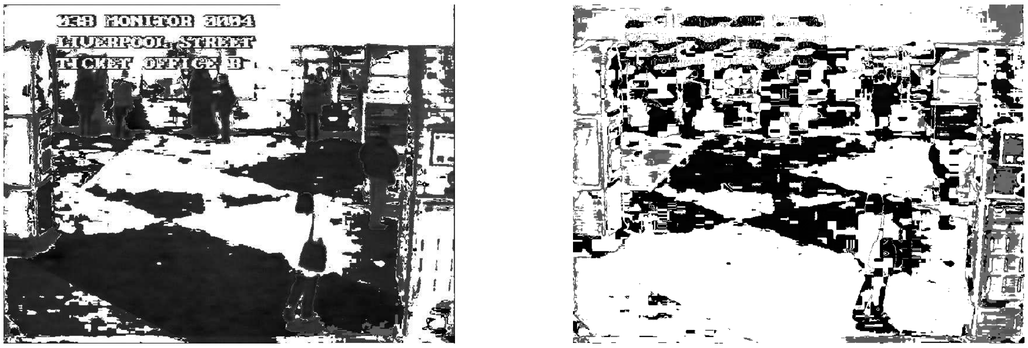


Figure 3.6: Chrominance channels C_B and C_R of the original Video Image (calculated according to CIE recommendation 601)

- It was recorded on analogue video tape (`s-vhs`) for algorithm development in the lab.
- After digitisation in the lab, the individual video images were JPEG compressed, compression ratio approx. 45 : 1.
- Some of the people in the image have a low contrast with the background. In these circumstances it is particularly important to have a good Difference Image to ensure that the areas occupied by people show up.

The video image shows compression artefacts from recording and JPEG compression. For example, the writing in the top left hand corner was originally white, now it is unevenly coloured, as can be seen in Figure 3.5. Also, the uniformly dark yellow floor area in the front shows green and red chequered “patches”, a typical effect caused by spatially resampling colour information into *compression blocks*.

Figure 3.6 makes the effects of compression clearer by showing the *chrominance* channels of the original Video Image. Both components show a low resolution, both spatially (large blocks of even shades) and precisionwise (few shades of grey). This is mainly due to analogue video signal compression (using composite video signals) and to digital image compression (JPEG), used for transmission of video images through the local computer network.

3.3.2 Local Image Filters Examined Here

Let us recall from Section 3.2.1 that in order to make edge detection more robust the image needs to be filtered such that people become more salient in the Difference Image. Local image filters F based on both linear and non-linear mappings f have been tested. The following mappings will be discussed:

Scaled Euclidian Norm

$$f : (R, G, B) \mapsto \sqrt{\frac{1}{3}} \cdot ||(R, G, B)||_2 = \sqrt{\frac{1}{3}(R^2 + G^2 + B^2)}, \qquad (3.3)$$

This is a straightforward and well-known method to map an element of $\mathbb{R}^3$ to $\mathbb{R}$. The scaling factor $\sqrt{\frac{1}{3}}$ makes sure that the range of values in the output is the same as in the input (i.e. 0..255). This mapping is the one used in the *Leeds People Tracker*[2]. However, there is no reason why this mapping should be particularly suitable for colour images; it would assume isotropic and uniformly distributed noise in the RGB colour channels which has no natural meaning in the “real world”.

Weighted Sum of $(\mathbf{R}, \mathbf{G}, \mathbf{B})$

The linear mapping

$$f : (R, G, B) \mapsto w_1 R + w_2 G + w_3 B \quad \text{with} \sum w_i = 1 \qquad (3.4)$$

is the general form of a number of useful mappings. Setting $w_1 = w_2 = w_3 = \frac{1}{3}$ yields the arithmetic mean of $\{R, G, B\}$. A more natural mapping would be onto the *luminance* Y of the pixel, that is, the perceived brightness. This can be done using the Y projection from *CIE* recommendation 709, denoted here as Y_{709} (taken from Ford and Roberts, 1998):

$$f : (R, G, B) \mapsto Y_{709}(R, G, B) = .2125\, R + .7154\, G + .0721\, B. \qquad (3.5)$$

The CIE has published other recommendations with different weights, depending on the hardware (camera, monitor) used in an application. However, the weights only differ slightly. This mapping completely disregards the colour information, which is an advantage because colour values are spatially sub-sampled and quantised by the JPEG image compression process.

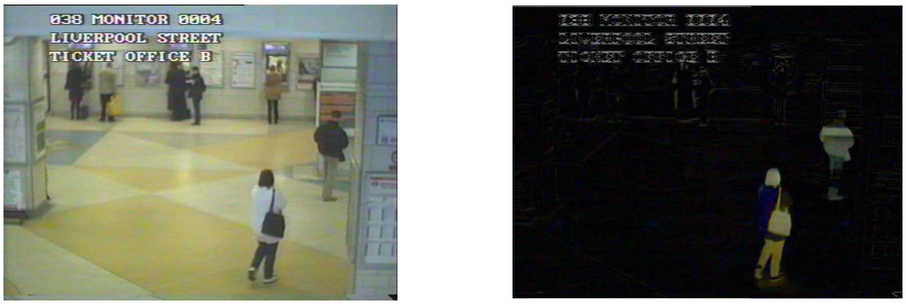
HSV-based Local Image Filters

Experiments were carried out with the mapping function f projecting (R, G, B) to the *hue*, *saturation* and *value* components, using an algorithm from Travis (1991, chap. 3). When differencing the *hue* components h_1 and h_2 of two pixels, the minimum angle between the two values is used:

$$\text{diff}(h_1, h_2) := 2 \cdot \min(|h_1 - h_2|, 255 - |h_1 - h_2|). \qquad (3.6)$$

---

[2] In the implementation of the *Leeds People Tracker* these values were not calculated explicitly but this is the equivalent operation.



(a) Video Image (b) Colour Difference

Figure 3.7: Video Image and Colour Difference Image used for Experiments

As with the other mappings, the scaling factor makes sure that the result takes almost all values in 0..255.

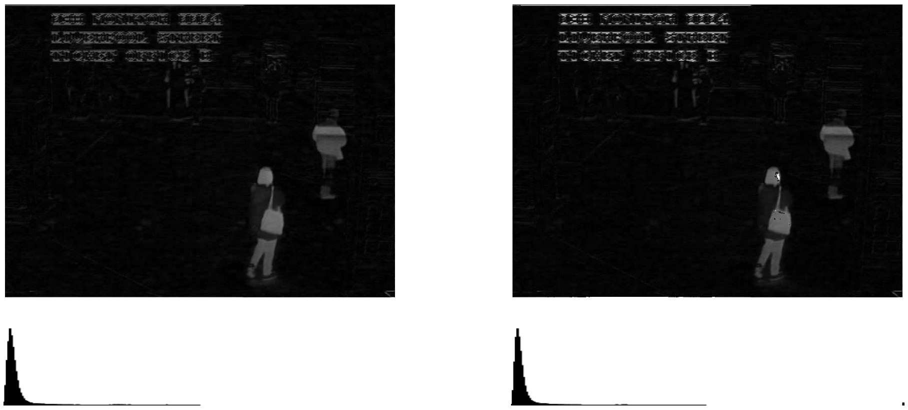
One important aspect to keep in mind when using and differencing HSV images is the well-definedness of its components. The *hue* and *saturation* components are not well-defined where *value* is close to 0 or 255 (i.e. for very dark and very bright pixels).

Figure 7 shows the Video Image and the Colour Difference Image used for the experiments presented here. In the following all greylevel images will be shown together with the histogram of their values.

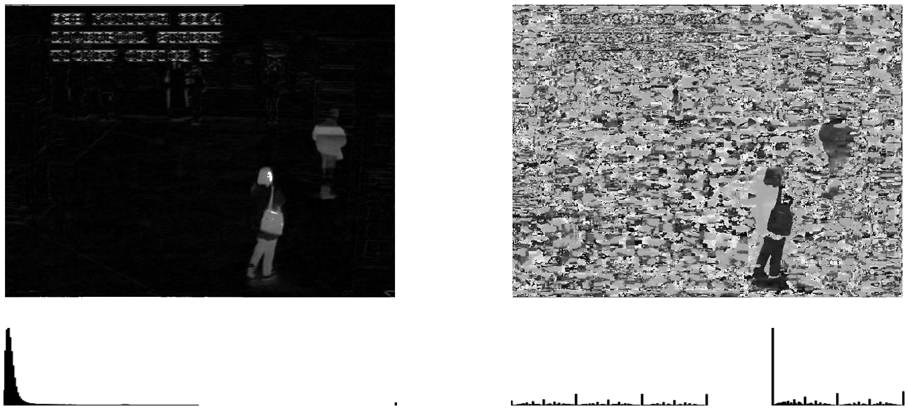
3.3.3 Method a: Filtering the Difference Image

The images in Figure 3.8 show the results for the mapping functions discussed earlier. The people in the back of the image have been static for a while and are therefore incorporated into the background. The focus is on the people moving in the foreground. They are are the woman in the white jacket (front right) and the man in the dark jacket leaving the image towards the right.

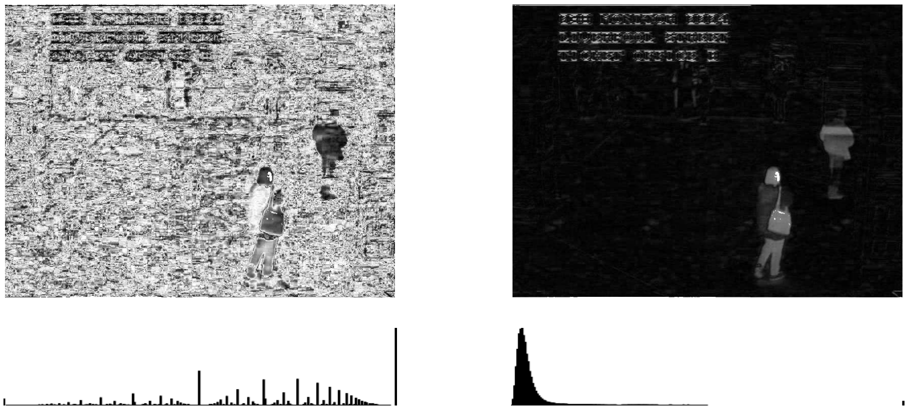
The results obtained by the Euclidian norm, arithmetic mean, luminance and HSV *value* mappings do not differ significantly. The images generated by the nonlinear HSV *hue* and *saturation* mappings, Figures 3.8(d) and 3.8(e), differ considerably from the other images. This is mainly because the *hue* and *saturation* components of an HSV image are not well-defined for where *value* is very large or very small—e.g. in dark regions, of which there are many in the Difference Image. Therefore these images are not useful for local edge detection. One further reason why these last two methods fail is the degradation of colour information through image recording and compression.



(a) Euclidian norm (b) Arithmetic mean



(c) Luminance Y_{709} (d) HSV: *hue*



(e) HSV: *saturation* (f) HSV: *value*

Figure 3.8: Filtering the Colour Difference Image

In the Colour Difference Image, shown in Figure 3.7(b), the woman's white jacket appears as dark blue after subtracting out the light yellow background. This creates difficulties for all local image filters. The HSV *value* mapping in Figure 3.8(f) and the Euclidian norm mapping in Figure 3.8(a) perform slightly better than the others. However, the resulting brightness of the jacket in the Difference Image is in no case sufficient for robust local edge detection.

As for the person to the right, the main parts of his body, especially the trousers, have many different shades of grey. Searching for edges around his shape outline is very sensitive to the edge detection thresholds introduced in Section 3.2.1, and this can cause difficulties.

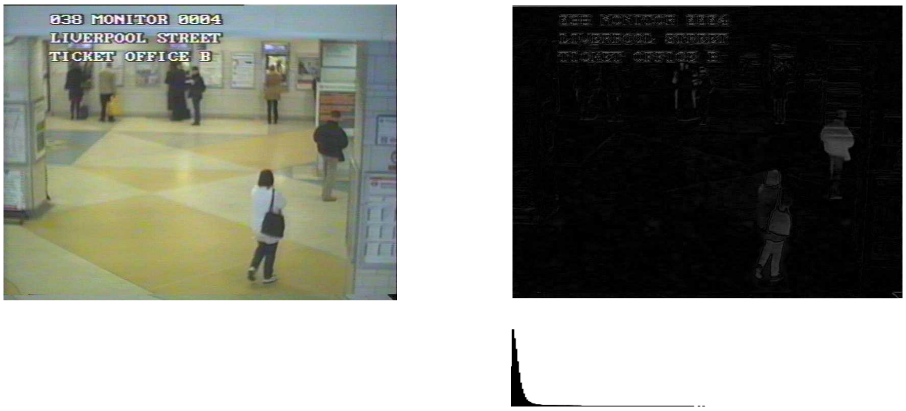
3.3.4 Method b: Differencing the Filtered Images

In a second stage of experiments both the Colour Background Image and the Colour Video Image were filtered using the same mapping function f, and the results were subtracted. The resulting Difference Images can be found in Figure 3.9.

With this *Method b*, the HSV *hue* and *saturation* components, shown in Figures 3.9(d) and 3.9(e), are for the most part of the image well-defined and show more reasonable results than before. This is mainly because they do not operate on the fairly dark Difference Image. However, they still cannot be used in the People Tracker.

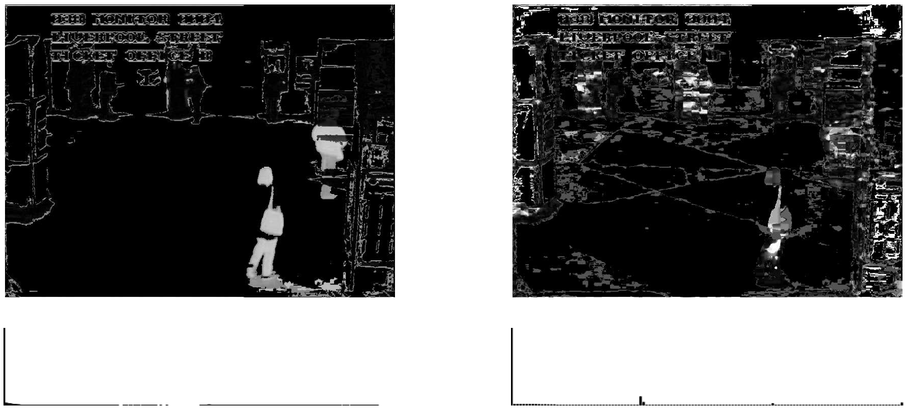
Of the remaining mappings, luminance and HSV *value*, Figures 3.9(c) and 3.9(f), give clearly better results than the Euclidian norm in Figure 3.9(b). Compared to the results with *Method a* there is not much change when it comes to the woman's coat. Other parts of both people in the front, however, appear much more salient, standing out clearly against the background. For edge detection this gives a significant advantage because it increases robustness by being less dependent on detection parameters such as threshold values. Looking at the right hand person's coat and trousers, these are now more uniformly shaded which means that edge detection is also less prone to being distracted by edges located inside the body.

The Best Result: Filtering with the luminance Y_{709} mapping using *Method b* gives the best Difference Image for People Tracking—see Figure 3.9(c). Comparing to the results with *Method a*, this local image filter also makes background edges stand out more clearly than in *Method a*. For global edge detection algorithms this could create difficulties. Our edge detector, however, only operates locally in the image so this does not pose a problem.


038 MONITOR 0004
LIVERPOOL STREET
TICKET OFFICE B

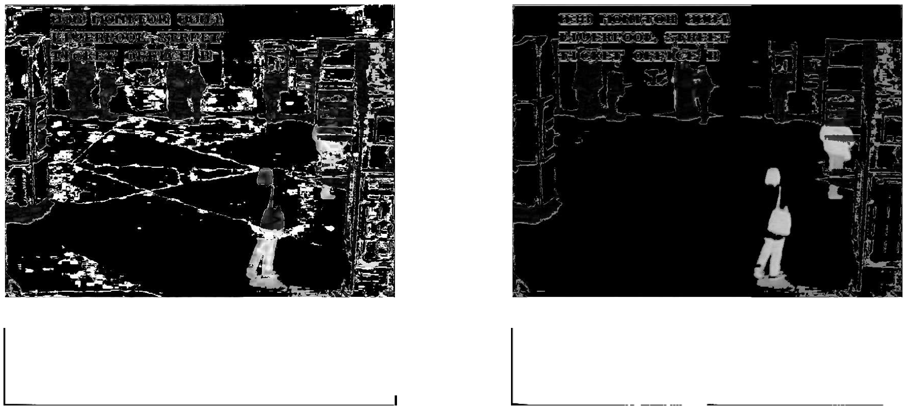
(a) Video Image

(b) Euclidian norm



(c) Luminance Y_{709}

(d) HSV: *hue*



(e) HSV: *saturation*

(f) HSV: *value*

Figure 3.9: Differencing filtered Video and Background Images

3.4 Discussion

This chapter has presented an approach to improve local edge detection for people tracking systems in order to increase robustness in the presence of image noise. The main problem in an integrated system like ours is the noise found in the *chrominance* channels of the image which bear all the colour information. This was seen in the images of the *chrominance* channels in Figure 3.6 on page 40.

The experiments show that the quality of any colour information in the video image is degraded to such an extent that it is no longer useful for local edge detection. Therefore, image differencing methods using only luminance (brightness) information give better results in local edge detection than methods involving colour information. Methods which filter out all colour information from the original images before any processing is done, for example *Method b* with a luminance Y_{709} mapping, provide the highest robustness to noise for using the People Tracker.

It should be noted that even in a system like ours, colour information can still be of good use for the identification of people once they are tracked.

Chapter 4

The Reading People Tracker

> In order for an automated visual surveillance system to operate effectively it must locate and track people reliably and in realtime. The Reading People Tracker achieves this by combining four detection and tracking modules, each of medium to low complexity. This chapter presents the new people tracking algorithms and the detailed structure of the tracker.

4.1 Designing a People Tracker for `ADVISOR`

This chapter describes the Reading People Tracker which works either standalone or as a subsystem of the *ADVISOR* system described in Section 2.3.1. The focus here is on tracking, specifically on how a number of detection and tracking algorithms can be combined to achieve robust tracking of people in an indoor environment.

Automated visual surveillance systems have to operate in realtime and with a minimum of hardware requirements, if the system is to be economical and scalable. As discussed in Section 2.1.3, this limits the complexity of models that can be used for detection and tracking. Any attempt at designing a People Tracker for a surveillance system like `ADVISOR` therefore has to consider the realtime aspect during algorithm design.

Figure 4.1 shows the overall system layout, with individual subsystems for tracking, detection and analysis of events, together with storage and human-computer interface subsystems to meet the needs of the surveillance system operators. Each of these subsystems is designed to run in realtime on off-the-shelf PC hardware, with the ability to process video input from a number of cameras simultaneously. The connections between the subsystems are realised by Ethernet. Images are transferred across the network using JPEG image compression. Other data, such as the output of the People Tracker and the results of the Behaviour Analysis, are represented in XML formats defined by a number of *XML Schemas* (XML Schema, 2001).

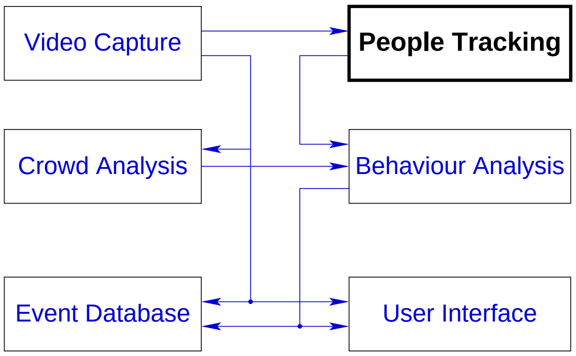


Figure 4.1: People Tracking as one of six subsystem of `ADVISOR`

The Reading People Tracker has been designed to run either as a subsystem of `ADVISOR` or in standalone mode. For its design four detection and tracking modules of medium to low complexity have been chosen, improved and combined in a single tracking system.

Originally based on the *Leeds People Tracker* introduced in Section 2.2.2, the most important one of the four modules is a slightly modified version of Adam Baumberg's Active Shape Tracker. The people tracker has been modified over time to increase tracking robustness, and adapted for use in `ADVISOR`. The tracker was ported from an `SGI` platform to a PC running `GNU/Linux` to facilitate economical system integration.

The tracking algorithms introduced in this chapter have been published in Siebel and Maybank (2002).

4.2 The Tracking Algorithm: Overview and Structure

The People Tracking subsystem is itself comprised of four modules which co-operate to create the overall tracking output, aiding each other to increase tracking robustness and to overcome the limitations of individual modules. As the general concepts and techniques of people tracking have already been described in Section 2.1, the following passages will focus on those aspects of the tracking algorithms which are special to this system.

4.2.1 Overview and General Features

Figure 4.2 shows an overview of how the four modules comprise the People Tracker. Before giving more detail on the individual modules, here is a brief overview of their

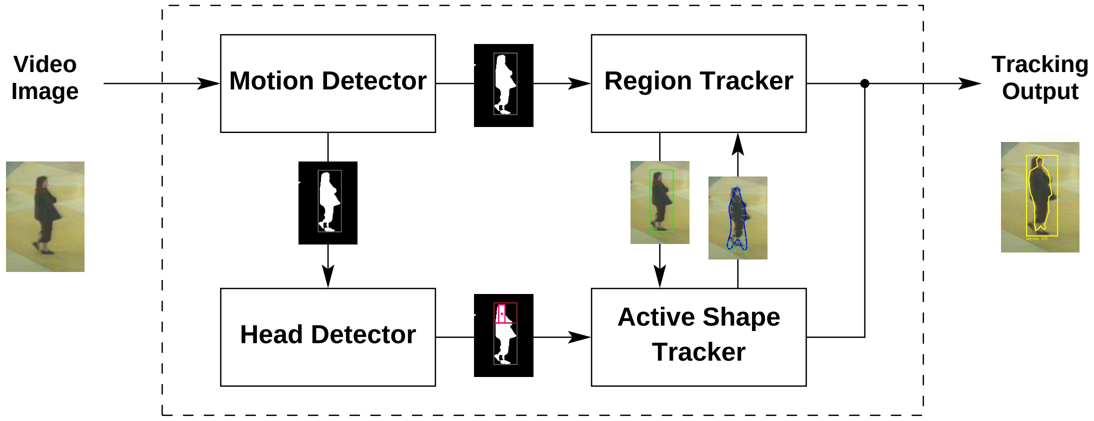


Figure 4.2: Overview of the four modules of the *Reading People Tracker*

functionality:

Motion Detector: This module models the background as an image with no people in it. The Background Image is subtracted pixelwise from the current video image and thresholded to yield the binary Motion Image. Regions with detected moving blobs are then extracted and written out as the output from this module.

Region Tracker: The Regions output by the Motion Detector are tracked over time by the Region Tracker. This includes region splitting and merging using predictions from the previous frame.

Head Detector: The Head Detector examines the areas of the binary Motion Image which correspond to moving regions tracked by the Region Tracker. The topmost points of the blobs in these region images that match certain criteria for size are output as possible positions of heads in these Regions.

Active Shape Tracker: This module uses an active shape model of 2D pedestrian outlines in the image to detect and track people. The initialisation of contour shapes is done from the output by the Region Tracker and the Head Detector.

The main goal of using more than one tracking module is to make up for deficiencies in the individual modules, thus achieving a better overall tracking performance than a single module could provide. Of course, when combining the information from different modules it is important to be aware of the main sources of error for those modules. If two modules are subject to the same type of error then there is little benefit in combining the outputs. The new People Tracker has been designed keeping this aspect in mind, and using the redundancy introduced by the multiplicity of modules in an optimal manner. These are the main features of the system:

- interaction between modules to avoid non- or mis-detection
- independent prediction in the two tracking modules, for greater robustness
- multiple hypotheses during tracking to recover from lost or mixed-up tracks
- all modules have camera calibration data available for their use
- through the use of software engineering principles for the software design, the system is scalable and extensible (new modules...) as well as highly maintainable and portable.

Each of the modules has access to the output from the modules run previously, and to the long-term tracking history which includes past and present tracks, together with the full tracking status (visibility, type of object, whether it is static etc). For each tracked object, all measurements, tracking hypotheses and predictions generated by different modules are stored in one place.

The following sections describes the four modules and their interactions in more detail. While reading through these sections, it might help to refer to Figure 4.4 on page 53 which details the complete tracking algorithm, and shows how the modules exchange data.

4.2.2 Module 1: The Motion Detector

Section 2.1.2 has introduced the *motion detectors*, and their application to detecting moving regions in the image. The Motion Detector used in this system is the *Reading Motion Detector* (see Section 2.2.2) which is also used in the *Leeds People Tracker*. The Motion Detector subtracts the background image from the current video image. Thresholding the resulting Difference Image yields the binary Motion Image from which moving regions are extracted.

4.2.3 Module 2: The Region Tracker

Figure 4.4 shows how the Region Tracker is connected to the Motion Detector. The Region Tracker tracks all moving regions detected by the Motion Detector. It uses a frame-to-frame region matching algorithm in which regions are matched according to their size and position. The new position of a tracked region in a given image is predicted using a first-order motion model for their movement in the previous image. The most important difference of this Region Tracker to existing methods (e.g. Brémond and Thonnat, 1997) is the temporal integration of static objects into the background. Further differences exist in the approach used for region splitting/merging, the way in which multiple hypotheses are generated and used, and in the cost function used to compare two regions.

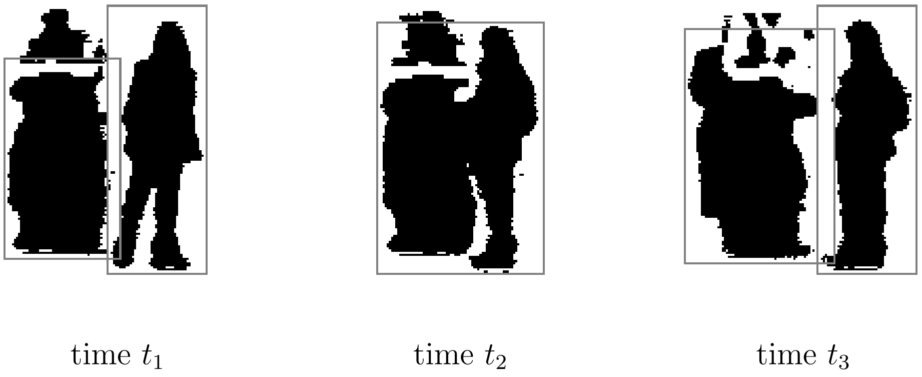


Figure 4.3: Problem during motion detection, solved by Region Splitting

The cost function for comparing two regions, e.g. one prediction and one measurement, in the matching process is as follows. Given a predicted region r_1 from the previous frame and a new measurement r_2, their matching score is calculated as a weighted sum of their difference in size and position of their centres in the image. Let Δ_x and Δ_y denote the absolute positional difference in x and y direction, Δ_w and Δ_h the absolute differences in width and height. Then the matching score $\delta(r_1, r_2)$ for regions r_1 and r_2 is given as

$$\delta(r_1, r_2) = \alpha_1 \Delta_x + \alpha_2 \Delta_y + \alpha_3 \Delta_w + \alpha_4 \Delta_h, \tag{4.1}$$

Good results have been achieved with $\alpha_1 = \alpha_2 = \alpha_4 = 1$, $\alpha_3 = 1.5$. The height of tracked objects is given an increased weight of 1.5 because it is generally more invariant under change of orientation.

Two special features of the Region Tracker will be explained in more detail: Region Splitting/Merging and the temporal integration of static regions into the background.

Region Splitting and Merging

In order to match the moving regions measured in the image by the Motion Detector to those predicted by the Region Tracker, the predictions for the current frame are combined to resemble the measured regions. If a combination of predicted regions matches the measurements, these predictions are adjusted and their track accepted.

This procedure becomes particularly important in cases like the one demonstrated in Figure 4.3 when one measured region, at time t_2, corresponds to two regions which were predicted at time t_1, $t_1 < t_2$: Two people, distinct at time t_1, come close to each other a little later, at time t_2, and overlap in the Motion Image. As a consequence, the Motion Detector extracts the two people as one moving region (the grey box in the middle). When at time t_3 the people separate again, the Motion Detector once more extracts two moving regions. These tracking and identification problems are overcome by Region Splitting by always trying to split regions into multiple tracked objects. Through this separation the two people can be tracked individually.

Due to the limited capabilities of our Motion Detector, detected moving regions are sometimes split in the image even if they are part of the same person. This is usually either due to partial occlusion or to low contrast of moving pixels with the background. Small adjacent regions detected by the Motion Detector are therefore merged before matching them to predicted regions.

Temporal Integration of Static Objects into the Background

One problem when tracking regions in the image is the overlap of two (or more) blobs arising from different people, as in the example in Figure 4.3. Assume that one of the corresponding objects is still moving and the other one has become static only a few frames previously. Then the Background Image usually does not include either of the two objects because it is not (and should not be) updated that fast. As a result, the Motion Detector still reports that both objects are moving. If these objects come close and overlap in the image, they are detected as one moving region which makes it difficult to maintain the correct identification of both objects. This can also generate difficulties for the Active Shape Tracker, which uses a pixelwise difference of Video Image and Background Image during shape fitting, because it has an incorrect notion of the background.

Static objects can be integrated into the background, thereby enabling correct detection and identification of objects moving nearby. The Motion Detector updates the background image periodically with the result that all static objects are eventually incorporated into the background. However, if static objects are incorporated too quickly into the background then it might not be possible to identify and track them when they start moving again. Moreover, there is the danger of detecting a "negative" of the object which remains stationary for a while and and then later moves on, because the background model now includes the object.

A simple procedure has been devised and implemented to incorporate static tracked regions *temporarily* into the background, thereby

- resulting in much better detection of movement in the vicinity of objects which have become static only a few frames previously, e.g. movement of people past stationary people,
- making it possible to restore the "empty" original background at the exact time a static object starts moving again,

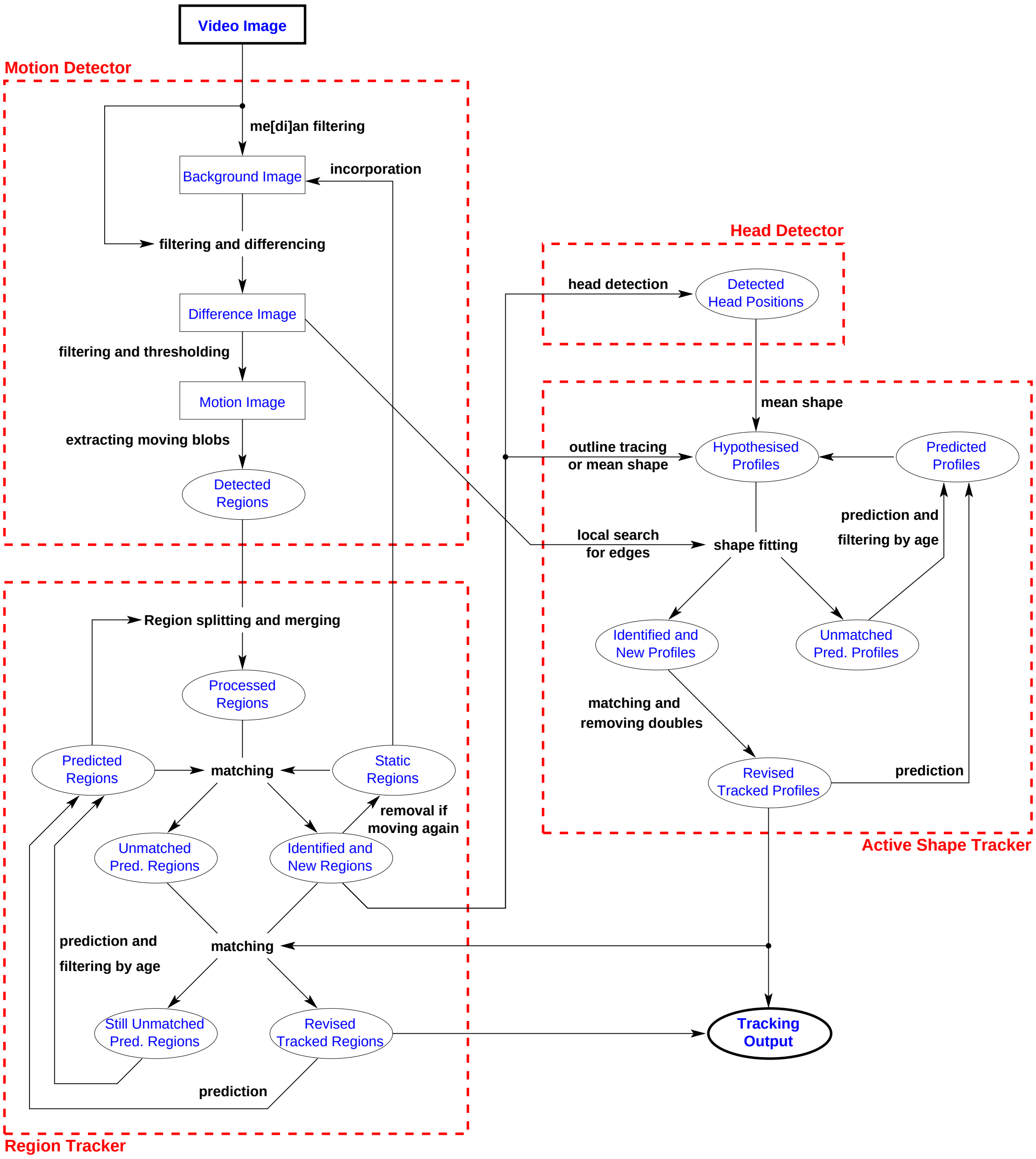

Video Image
Motion Detector
me[di]an filtering
Background Image
incorporation
filtering and differencing
Difference Image
filtering and thresholding
Motion Image
extracting moving blobs
Detected Regions
Head Detector
head detection
Detected Head Positions
mean shape
outline tracing or mean shape
Hypothesised Profiles
Predicted Profiles
prediction and filtering by age
local search for edges
shape fitting
Identified and New Profiles
Unmatched Pred. Profiles
matching and removing doubles
Revised Tracked Profiles
prediction
Active Shape Tracker
Region splitting and merging
Processed Regions
Predicted Regions
matching
Static Regions
removal if moving again
Unmatched Pred. Regions
Identified and New Regions
prediction and filtering by age
matching
Still Unmatched Pred. Regions
Revised Tracked Regions
prediction
Tracking Output
Region Tracker

Figure 4.4: The Algorithm of the *Reading People Tracker*

- enabling the identification of the object starting to move again, by keeping a copy of the necessary data (position, size, identity record).

The areas in the image where static objects have been detected and incorporated into the background are specially marked so that the background modelling algorithm does not include these objects into the Background Image.

Multiple Hypotheses For Matching

When matching the moving regions measured by the Motion Detector to those regions predicted by the Region Tracker, multiple hypotheses are used. The predictions of all currently tracked regions form the first set of hypotheses. As presented above, these original hypotheses are combined to form new hypotheses which approximate the measurements. These new hypotheses are marked as being created by splitting and added to the original hypotheses. If the measurements from the Motion Detector match a hypothesis which was created by combining predicted regions, these predictions are accepted. However, if an original prediction seems to fit the measurements better, that one is accepted. Choosing the optimal track uses an adapted version of the cost function given in (4.1) above,

$$\delta_p(r_1, r_2) = \alpha_1 \Delta_x + \alpha_2 \Delta_y + \alpha_3 \Delta_w + \alpha_4 \Delta_h + \mathrm{p}(r_1, r_2), \tag{4.2}$$

where $\mathrm{p}(r_1, r_2)$ is a function which yields a penalty for static regions[1], 0 otherwise. Thus a higher cost is given for static objects such that a person walking by a static person does not get matched to the static person too easily.

4.2.4 Module 3: The Head Detector

The W^4 system by Haritaoglu et al. (2000) introduced in Section 2.2.3 looks for vertical peaks in the motion image to detect heads of moving people. Inspired by this idea a simple algorithm has been implemented to detect head positions in moving regions tracked by the Region Tracker. Given a region r, the Head Detector works as detailed in Algorithm 4.1. The main idea of the algorithm is to create something similar to a vertical projection histogram of the upper part of every tracked region. Peaks in this vector of heights v correspond to possible head positions. Isolated pixels (e.g. noise) and “holes” in the Motion Image are ignored when creating v. While scanning v for peaks, camera calibration is used to check the size of these peaks.

Figure 4.5 shows the region r outlined in grey with the search area $\tilde{r}$ outlined in dark red and the detected head position h_1 marked in magenta. The set $\{h_i\}$ of head

---

[1] At most one of the two regions r_1 and r_2 is static because one of the regions is a new measurement or other hypothesis.

Given: A region r which is tracked by the Region Tracker.

1. Select the upper part of r, $\tilde{r}$.
2. If $\tilde{r}$ is too small to contain a head, stop with no result.
3. Create a vector v of "heights" in the image $\tilde{r}$:
    (a) For each column c in $\tilde{r}$, examine the pixel values from the top until reaching a non-isolated pixel which is classified as moving.
    (b) The distance of this pixel to the bottom of the image $\tilde{r}$ is stored in v as the person's "height" at column c.
4. Scan v for intervals $v_i \subset v$ with the following properties:
    (a) The values within v_i do not differ significantly.
    (b) The width of v_i is a possible width of a head in the image.
    (c) The histogram values in $v \setminus v_i$ which are near to v_i are significantly lower than the values inside v_i.
5. For each v_i found in step 4 define the point h_i in the region $\tilde{r}$:
    (a) Let the x coordinate be the centre of v_i
    (b) Let the y coordinate be the mean of the highest and the lowest value in a small neighbourhood of v around v_i.
6. Stop with result head positions $\{h_i\}$.

Algorithm 4.1: Head Detection

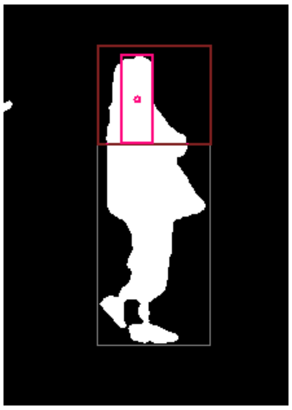


Figure 4.5: Head detected by the Head Detector

positions is stored together with the associated region r for later use by the Active Shape Tracker.

The algorithm of the Head Detector is optimised for speed not accuracy. It is much simpler than standard face detection and tracking methods which use more sophisticated means such as skin colour models (Crowley and Bérard, 1997; Oliver et al., 1997). The purpose of the Head Detector in our system is to aid in the initialisation and validation of tracks by the Active Shape Tracker. Wrongly detected heads (false positives) usually do not influence tracking output because no person can be detected for this hypothesised head. The detected heads themselves are not part of the final tracking output; they are just temporary results. Owing to its design and implementation the Head Detector is very fast, adding no noticeable overhead to the overall CPU time used by the People Tracker—an important aspect in a realtime system.

4.2.5 Module 4: The Active Shape Tracker

Moving regions detected and tracked by the Region Tracker are examined more closely by the fourth module, the Active Shape Tracker. It uses the parameterised *B-spline* model of a generic pedestrian outline shape. Most of the functionality developed in the *Leeds People Tracker* (see Section 2.2.2) has been preserved in our system, while addressing the most important deficiencies, as established in Section 2.2.2. This was achieved mostly by adding a Region Tracker and a Head Detector to the system and using their detection and tracking results:

Improved Generality: The model used in the Active Shape Tracker, like the one in the *Leeds People Tracker*, is trained to recognise walking pedestrians. When this assumption on people's appearance breaks down, e.g. due to occlusion, low contrast in the image or when they sit down, the Region Tracker will try to keep tracking the person, so the overall track and identity of the person is not lost.

Better Initialisation of Tracks: The Active Shape Tracker now uses the output from the Region Tracker and the Head Detector to initialise tracks. This usually results in multiple hypotheses of outline shapes for the same person which are compared and matched by an improved matching algorithm. Only the best hypothesis for each person's shape is retained.

Handling Occlusion and Low Contrast: In the presence of severe occlusion and/or low contrast in the image the Active Shape Tracker can either lose track or fail to initialise a track. In these cases the Region Tracker will still try to track the person, with the aim of preserving the person's identity and long-term tracking history (movement within the station). Once the Active Shape Tracker is able to take up the track again the full tracking history is thereby restored.

Handling Lost and Re-gained Tracks: When the Active Shape Tracker loses track of a person and re-gains it at a later stage, it uses the tracking data of the Region Tracker if that succeeds in keeping track of the person. Thereby the correct identity and tracking history is assigned to the re-gained track. This works in a similar way if the Region Tracker loses track and the Active Shape Tracker keeps tracking an object.

Additionally, both the Active Shape Tracker and the Region Tracker now keep predicting the position of a lost track for a very short time after the track is lost. This makes up for the problem where a track is lost for a frame or two only, and re-gained afterwards.

Increased Maintainability: The People Tracker has been ported from `SGI` to an economical `GNU/Linux` based PC, and then fully re-engineered. The code is now highly maintainable, and documented in a 45-page reference manual (Bouffant et al., 2002). Details of the re-engineering process and new software characteristics can be found in Chapter 5.

Handling Changes in Shape: This aspect of the *Leeds People Tracker* has not been changed in our Active Shape Tracker. However, by using a Region Tracker in parallel to the Active Shape Tracker the chance to lose a track because the shape fitting process does not converge has been reduced.

4.3 Module Interaction and Other Features

The work on the Reading People Tracker presented here covers the co-operation between modules to produce robust people tracking functionality, as well as software engineering aspects such as system integration and its maintainability characteristics (e.g. extensibility and scalability). Most of the maintainability issues have been the subject of two case studies which are presented in chapters 5 and 6. In the following sections of this chapter, the focus will therefore be on the remaining aspects.

4.3.1 Interaction Between Tracking Modules

Figure 4.6 shows the data flow between the four detection and tracking modules, as well as the Background Model and the Tracking Status and History database. The modules are operated in the order:

1. Motion Detector,
2. Region Tracker,
3. Head Detector,
4. Active Shape Tracker.

Each of the modules has access to the output from the modules run previously, and also to the long-term tracking history which includes past and present tracks, together with the full tracking status (visibility, type of object, whether it is static etc). For each tracked object, all measurements, as well as tracking hypotheses and predictions generated by different modules are stored in a central place. As a result, a module can access all tracking data generated by other modules for any tracked object.

Module 1: The Motion Detector.

The Motion Detector uses the background model to detect new objects. With the help of the Region Tracker static objects are quickly incorporated into the background, thereby enabling the correct detection of other objects moving in front of the static objects. The background model is kept up to date by removing objects from it if they begin to move. This, in turn, helps the Region Tracker to track objects more reliably by improving the Motion Detector's measurements.

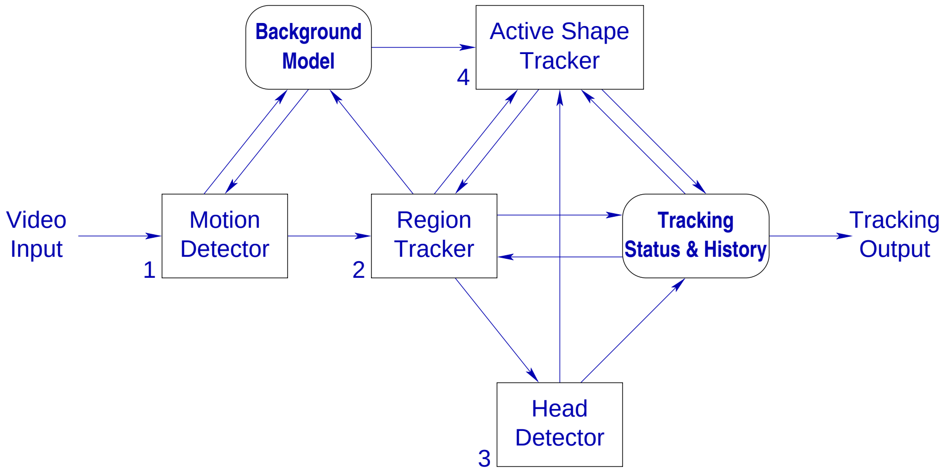


Figure 4.6: Interaction between tracking modules. The numbers refer to the sequence in which the modules are operated in the tracking stage.

Module 2: The Region Tracker.

The Region Tracker uses the Tracking Status and History database to track regions over time. It creates multiple hypotheses of tracks using Region Splitting and Merging, and keeps track of objects even if they are not visible in some frames. If the Region Tracker is unsure about a track or it cannot detect an object, it consults the Active Shape Tracker's output for that object. If the associated shape model of a person is successfully tracked, the Region Tracker uses the bounding box of the tracked shape outline as a new hypothesis for the region's position and size.

Another way the Region Tracker uses the output of the Active Shape Tracker is to split up large regions if they contain more than one person. After both the region and the associated person shape(s) have been tracked, the Region Tracker checks whether a significant part of the region was not covered by the shape(s) contained in the region. This situation occurs when two or more people are close together and detected as one region, for example when they enter the scene in a group, one person occluding the other. Once the Region Tracker has established that there is more than one person in the region, the region is divided into two and each subregion tracked separately in order to establish the tracks of all people within the group. Camera calibration is used in this process to determine whether the remainder of the region, after splitting it, is large enough to contain more people, and how many there might be. In subsequent frames, the Region Tracker uses Region Splitting to correctly split up the output from the Motion Detector into two or more regions in order to track every person correctly. These split-up regions are then processed separately by the Head Detector, and the Active Shape Tracker tries to initialise person outline tracks in each of them.

Module 3: The Head Detector.

The Head Detector uses the regions detected and tracked by the Region Tracker. It stores a list of candidate head positions together with the associated regions in the Tracking Status database. These head positions are mainly used by the Active Shape Tracker.

Module 4: The Active Shape Tracker.

The Active Shape Tracker has the largest number of inputs from other modules. The most important support the Active Shape Tracker receives from other modules is in the initialisation and identification of tracks. The initialisation refers to the process of estimating the position and size of an outline shape in the image. Once a track is initialised the Active Shape Tracker uses its own predictions and tracking status, stored in the central Tracking Status and History database, to keep track of a person.

The initialisation of a track utilises detection results both from the Region Tracker and the Head Detector. In addition to the initialisation of shapes from tracked regions, the heads detected in a region are used to determine possible positions of people. In this process, camera calibration is used to create hypotheses of the most probable size and position of a person in the image. This initialisation by head positions is particularly important when there is more than one person in a given region, e.g. when a group of people is detected as a single moving region.

Additional attempts to initialise tracks are made for regions and already tracked outline shapes if the Active Shape Tracker detects that a shape is too large or too small to be a person. This situation can occur when the upper or lower part of the person is occluded. Using camera calibration, two additional hypotheses are created for the tracked object to cover the cases that either the lower or the upper part of the person's outline is visible. Hypotheses created in this way are added to the tracked object for post-processing and filtering, described in Section 4.3.2 below.

During the shape fitting process, the Active Shape Tracker also uses the Difference Image to facilitate local edge search around the current shape. In this way, the Active Shape Tracker benefits significantly from the temporal integration of static objects into the background by the Region Tracker, resulting in more accurate tracking results.

When a new track is initialised by the Active Shape Tracker, it is assigned the identity of the associated region. This is especially important in cases when a track is lost by the Active Shape Tracker, e.g. due to occlusion. If the Region Tracker keeps the track then the Active Shape Tracker can re-establish the identity of the tracked person when the track is re-gained at a later time.

4.3.2 Hypothesis Refinement

After running all four detection and tracking modules, the data and tracking hypotheses generated by them are further analysed and filtered. The trackers usually generate more than one hypothesis for each tracked person, and the information can be of different types (moving region, head position, shape model). In order to reduce the number of hypotheses, they are first pairwise compared to see whether they are multiple observations of the same object. Those which are, e.g. multiple shape models for the same person, are further compared using a track confidence measure generated by the tracker and the positional uncertainty for the predicted position. The better one of the two tracks is then accepted as valid and the other one discarded, whilst making sure that the original identity of the person or object is carried over to the next frame.

Multiple hypotheses of the same track are kept if they are considered as possibly valid. Although only the best tracks appear in the tracking output more tracks are kept in the Tracking Status database, and predictions are made of these by the associated trackers. This way, a hypothesis not matched in one frame, e.g. in the event of a partial occlusion or a missed (dropped) frame in the system, is not lost but may again be matched to measurements and the track re-acquired, at a later time.

4.3.3 Software Engineering Aspects

The People Tracker presented here is an extension, in functionality and application, of the *Leeds People Tracker* introduced in Section 2.2.2. Before adding new modules, the People Tracker was completely re-engineered, yielding a new design. The new software is highly maintainable and portable, and a software process for all maintenance work is well defined and documented. The source code now strictly adheres to the ISO `C/C++` standard, compliant with ISO/IEC 9899 (1999), as well as POSIX 1003.1c-1995 (1995) extensions for all multi-threading functionality, making it easily portable. While the code is being maintained under `GNU/Linux`, it also compiles under `Microsoft Windows 2000`.

Extensibility and scalability were also kept in mind while designing the new tracker. For instance, all classes handling the data and tracking functionality for one camera (i.e. one video input) are encapsulated within one `Camera` class. As a result, the handling of multiple cameras simply amounts to creating multiple instances of the `Camera` class. The original version of the software handled functionality such as the video input, tracking etc directly in the `main()` program, and only one video input was implemented. On a larger level, all instances of the `Camera` class within one area of the underground station are contained within one `PeopleTracker` class. Hence the system design allows for scaling of the system simply by creating more then one instance of the `PeopleTracker` class. This was done to achieve a high level of scalability, although this functionality is not required at present.

The design of the People Tracker allows for the easy addition of new tracking modules for extending the tracker or adapting it to new application areas. Existing tracking modules can be enabled and disabled, depending on the application and circumstances (e.g. available CPU time).

More software engineering aspects, e.g. the avoidance of global functions and the use of assertions in the implementation, are covered in Chapter 5.

4.4 Demonstration and Discussion

The tracking algorithms of the Reading People Tracker is demonstrated and discussed below. The validation of the tracker can be found in Chapter 7.

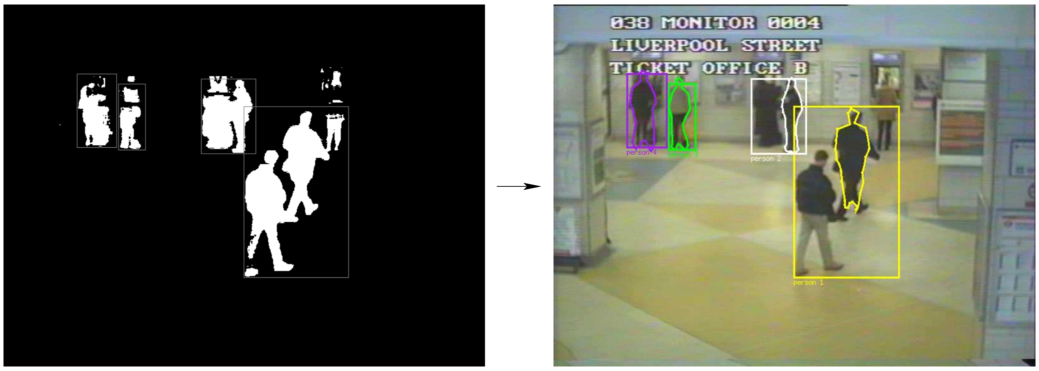
4.4.1 Testing on a London Underground Sequence

To demonstrate the robustness obtained by combining tracking modules, several experiments were performed with video sequences from surveillance cameras. A video sequence from a surveillance camera in a London Underground station has been chosen for the following demonstration because it has many potential difficulties for people tracking:

- stationary people waiting in queues at the three counters of the ticket office,
- people who occlude each other as they walk past,
- low contrast of some people to the background,
- people in groups who come close to each other and part again.

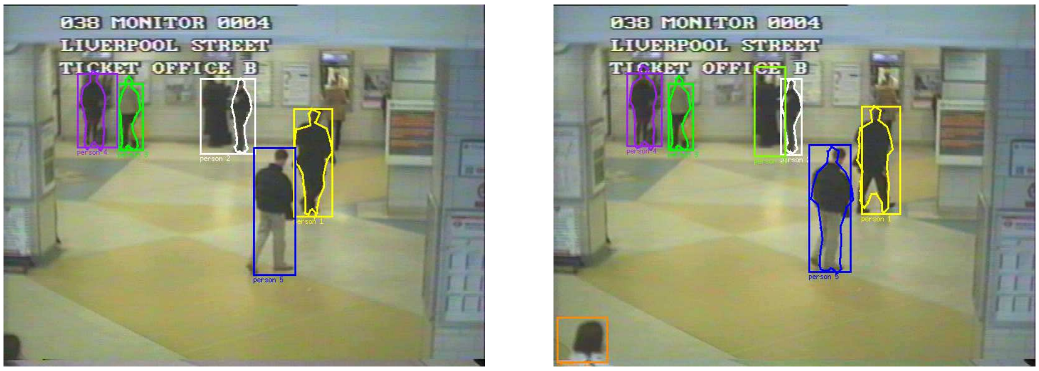
The sequence was digitised at full PAL resolution (768 × 576 pixels) at a frame rate of 5 frames per second (fps). The People Tracker runs in realtime (that is, 5 fps) with the number of objects shown (up to 9 objects), on a 1 GHz `Intel Pentium III` based PC, running under `GNU/Linux`. The computing time includes software JPEG decompression of the video input and annotation output in *XML*, but no display (this is done by the Human Computer Interface in the `ADVISOR` system). Screen output, when enabled during development, approximately doubles the amount of overall CPU time used by the People Tracker, because no optimisations were applied to graphics routines.

Figures 4.7 and 4.8 show the tracking output of the tracker. Regions are identified by their defining bounding boxes, and associated shapes from the Region Tracker are



(a) Frame 2, Motion Image

(b) Frame 2, Video Image



(c) Frame 3

(d) Frame 4

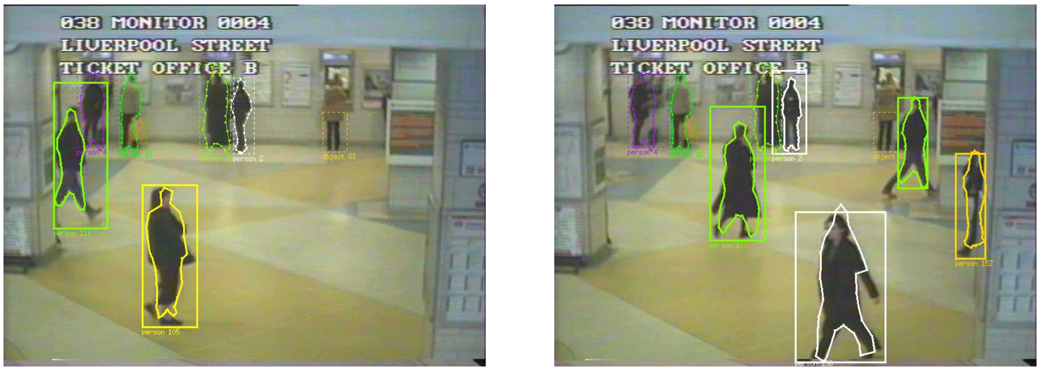
Figure 4.7: Frame 2 (Motion and Video Images); Frames 3 and 4 (Video Images).

drawn in the same colour. In the following it will be examined how the new tracking modules work in practice.

In the motion image of **Frame 2**, Figure 4.7(a), some regions contain more than one person. The problem arises because the people concerned are close together, and no previous tracking data is available. As a consequence, not all shapes were initialised correctly by the Active Shape Tracker, as can be seen in the video image, Figure 4.7(b), where the tracking results are shown. We notice, however, that by using the Head Detector, one person's shape was initialised correctly in each of the two large regions (track 1, yellow box, and track 2, white box).

In the Hypothesis Refinement stage, the Region Tracker tries to separate the detected person within the large regions from the remainder of the region. **Frame 3**, Figure 4.7(c), yields the new track of the person in the middle (track 5, blue box).

In **Frame 4**, Figure 4.7(d), the track of region and outline shape of person 2 (white) are aligned, enabling the separation from the adjacent person to the left. The track of the newly detected left hand person is picked up by the Region Tracker (track 10, light green box) and tracked using Region Splitting. When the Active Shape Tracker also starts tracking the person a little later, their identity and track history are already established.



(a) Frame 39

(b) Frame 55

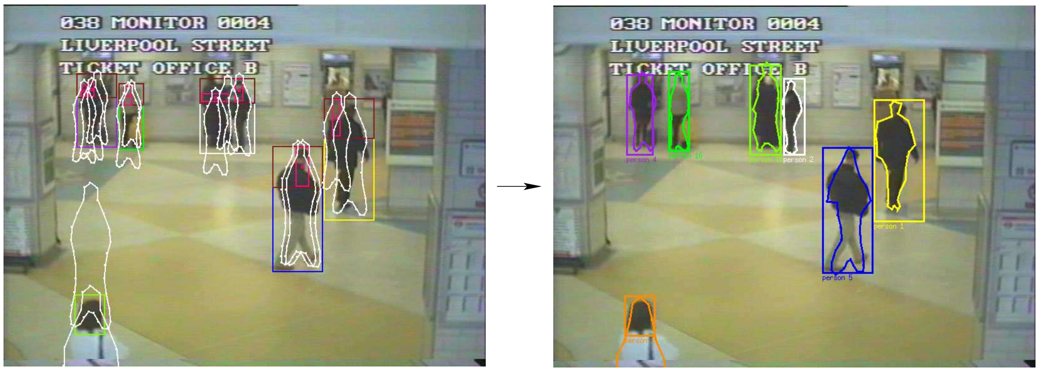
Figure 4.8: Frames 39 and 55 (Video Images).

Frame 39, Figure 4.8(a), a few seconds later, shows that the people at the counter have been detected as static and temporarily incorporated into the background. This is indicated by a dashed outline around the static objects. The person at the right counter cannot be tracked by the Active Shape Tracker because their jacket does not stand out against the background. However, the Region Tracker keeps track of their dark trousers until the full outline can be tracked by the Active Shape Tracker.

In **Frame 55**, Figure 4.8(b), Person 2 (white) has started to move again. Their image is removed from the background and both Region Tracker and Active Shape Tracker continue tracking with the correct identity. Meanwhile, a person has walked past all five static people without their track being lost.

4.4.2 Analysis

In the following analysis the focus will be on the way in which the new tracking system deals with the most important shortcomings of the original *Leeds People Tracker* discussed in Section 2.2.2.



(a) Initialisation of new tracks in the Active Shape Tracker

(b) Final tracking output after hypothesis refinement

Figure 4.9: Initialisation and refined tracks, frame 6

Initialisation of Tracks

Figure 4.9(a) shows the locations in the image where the Active Shape Tracker looks for people, in order to initialise new tracks. Initial shape estimates are projected onto the image before the shape fitting process is started. Hypotheses of already tracked people are also used, but they are not displayed here. Some of the shapes examined in the initialisation process are very close to each other, and in the shape fitting process they converge to the same person. In the Hypothesis Refinement stage, only the strongest hypotheses are kept and the others are abandoned.

Potential head positions as detected by the Head Detector are marked in the image in red, showing search regions and estimated head positions. It can be seen that most of the heads are detected, and an active shape model is initialised in the vicinity of every person in the image.

The bottom left corner of Figure 4.9(a) shows a detected region from the Region Tracker, in this case the head of a person. The Active Shape Tracker uses camera calibration to establish that this region is too small to be a person and examines two hypotheses: either the region constitutes the head of a person or their feet[2].

Figure 4.9(b) shows the final tracking output for the same frame shown in Figure 4.9(a). Seven out of eight people have been correctly detected and are tracked. At the rightmost counter stands a man whose beige jacket has a particularly low contrast to the background. Additionally, most of his body outline is occluded by the person in front of him. This is why the Active Shape Tracker does not find enough edges during the shape fitting, and consequently does not detect him. The

---

[2] This feature was also part of the original *Leeds People Tracker*.

Region Tracker also cannot pick up the track because due to the low contrast, only very few pixels of the visible part of the person show up as moving in the Motion Image.

Dealing with Occlusion and Low Contrast

The new tracking system handles occlusions better than the *Leeds People Tracker* in the following ways.

- Static objects are temporarily and quickly incorporated into the background. This feature of the Region Tracker (and the Motion Detector), as discussed in Section 4.2.3, helps to avoid problems with people overlapping in the image. This was seen in Frame 39 above, Figure 4.8(a) on page 64, when a person was tracked even though they walked past other people who had been stationary for a very short period of time.

- The tracks of individual people are not lost so frequently because there is now a Region Tracker linked with the Active Shape Tracker. The two modules track the person shapes and associated regions independently and their tracking results are combined. Occlusion still affects the Active Shape Tracker when a large part of a person's outline is not detectable. The Region Tracker, due to its nature, does not observe the outline and so it is less affected by occlusions of the outline. Region Splitting and Merging also helps the Region Tracker to avoid problems of occlusion.

Limited Generality

Through the combination of the Region Tracker and the Active Shape Tracker problems with the *Leeds People Tracker's* limited generality are overcome. If the Active Shape Tracker loses track of a person because their shape cannot be recognised or explained by the pedestrian shape model, the Region Tracker will keep tracking the person. The Reading People Tracker is therefore less likely to lose track of a person e.g. when they sit down or there is low contrast. An example was seen in Frame 39 above, Figure 4.8(a) on page 64, where the Region Tracker tracks the lower part of a person (trousers) while the upper part is not detectable and the Active Shape Tracker therefore fails initialise the track.

4.4.3 Remaining Problems

The following situations still pose a problem for the Reading People Tracker:

Heavy Occlusion: The strategy of the Region Tracker and the Active Shape Tracker is to separate people who overlap in the image, and track them individually. In the presence of crowding or in large groups of people who overlap in the image this might not be possible.

Lighting Changes: The Motion Detector cannot handle rapid changes in lighting conditions. While the system is designed to work indoors with constant artificial lighting, there are some situations in which the level of lighting changes. For example, when a train arrives in the platform the camera's automatic gain control might adjust to the new overall brightness of the scene. Thereby other objects might appear darker or lighter in the image while they really have not changed. This problem could be solved by replacing the Motion Detector by a better one. Currently the restrictions on CPU time are too stringent to allow for this.

Recognising Re-gained Tracks: If a track is lost and re-gained at a later stage, only a very simple comparison is done with old tracks to try to identify the person. Re-identification is based on predicted motion. Therefore it is only possible if the track has been lost in the past few frames and the person has not moved irregularly. An appearance-based model of every tracked person could help to identify people under more complex conditions, e.g. when they leave the field of view for a while and then come back later.

4.4.4 Summary

The Reading People Tracker has been developed to be part of the integrated visual surveillance system `ADVISOR` for operation in underground stations. The People Tracker is based around an Active Shape Tracker which tracks the 2D outlines of people in video images. In order to achieve robust tracking of people, a Region Tracker and a Head Detector were added to the original tracker design. interaction

By combining multiple modules to form one People Tracker, and fusing their output to generate the tracking output, a higher tracking reliability is achieved than any of the individual trackers can provide on its own. The Region Tracker with its Region Splitting and Merging, as well as its Temporal Background Integration features plays an important role, helping when occlusion creates difficulties for the Active Shape Tracker, and in the identification of tracks for correct long-term tracking. The Head Detector provides additional measurements to the Active Shape Tracker to aid in the initialisation of tracks.

This tracking performance is achieved while keeping the hardware requirements to a minimum; the tracker runs in realtime on off-the-shelf hardware. Its software structure enables the Reading People Tracker to work with input from multiple cameras, and to be further extensible and scalable.

The Reading People Tracker will be validated in Chapter 7.

Chapter 5

Maintainability of the Reading People Tracker

Adapting the People Tracker for use within the integrated system `ADVISOR` required fundamental changes, due to new requirements and close integration with other components. This chapter describes the corrective re-engineering measures that rendered the software highly maintainable, making the necessary changes possible and easy to carry out.

5.1 Introduction

5.1.1 Software Maintenance Techniques

Software maintenance is the modification of a software product after delivery. It is classified into four categories (Fenton and Pfleeger, 1996, p. 354–355):

1. *Corrective maintenance* refers to modifications for correcting problems in an implementation.
2. *Adaptive maintenance* refers to modifications for adapting a product to changed environments, both software and hardware.
3. *Perfective maintenance* refers to enhancements such as making the product faster, smaller, better documented, creating a cleaner structure, and adding new functionalities because of new user requirements.
4. *Preventive maintenance* involves changes aimed at preventing malfunctions and improving maintainability of the software.

The maintainability of a piece of software is determined not only by the state of its source code, but by a variety of other factors. In this context it is useful to define the following term.

5.1. Definition (SOFTWARE ARTEFACT)**:** *A software artefact is the result of any activity in the software life-cycle, such as requirements, architecture models, design specifications, source code and test scripts.*

These *software artefacts* are the determining factors for the maintainability of a piece of software. Initial design of software should aim at developing a product so that all necessary software artefacts exist and exhibit a high level of quality. Furthermore, these artefacts should be consistent. For example, any design specification should match its implementation. This way, the above categories of maintenance activities become easy to perform in terms of time and effort—that is, the software has a high level of *maintainability*. It often happens, however, that modifications are done without following proper software engineering principles, with the effect that the software maintainability decreases. This makes further maintenance more and more costly in terms of both effort and time.

Consider the example of cohesion and coupling (Fenton and Pfleeger, 1996, p. 309–317). An ideal implementation should maximise the interaction between various elements within a class (cohesion) and minimise the interaction across classes (coupling). However, if too much "patch work" is done on a haphazard basis these characteristics can easily be lost. As a result, the implementation no longer satisfies the desired levels of cohesion and coupling. For better maintainability, it is also expected that the various artefacts of software—from requirement document through to design—remain consistent in relation to each other. In many cases this consistency between various artefacts is lost over time because of the bad execution of the maintenance process.

In such a case, it may become necessary to re-organise the software with the aim to restore the system's maintainability. This approach is often called *re-engineering* (see, for example, Weide et al., 1995). Re-engineering is characterised by the following objectives:

- to generate the software artefacts from the implementation and any existing documents,
- to bring these artefacts into a consistent state, possibly through re-design or re-specification. The re-design should aim at restoring the quality characteristics of the software (see ISO/IEC 9126, 1991, for a definition of these); and
- to re-structure the implementation so that it reflects the new design.

Thereby, the system's maintainability is restored. Re-engineering often involves *reverse engineering*. *Reverse engineering* is the process of understanding and modifying software systems. It involves identification of the components of an existing system and their relationships. Furthermore, it also aims at creating high level descriptions of various aspects of the existing system (Waters and Chikovsky, 1994; Weide et al., 1995).

The primary challenge with regards to re-engineering is to understand the existing software along with its associated artefacts. In order to accomplish this, one needs two kinds of information: *static information* and *dynamic information*. *Static information* describes the structure of the software in relation to the source code, while *dynamic information* describes its run-time behaviour (Richner and Ducasse, 1999).

Refactoring (Fowler et al., 1999) is one technique to correct design flaws in object oriented systems. Refactoring operations reorganise a class hierarchy by shifting responsibility between classes and redistributing instance variables and methods. Demeyer et al. (2000) discuss heuristics for identifying various types of refactoring operations applicable to a piece of software. The authors deal with the following three types of refactoring:

1. splitting methods into smaller chunks to separate common behaviour from the specialised parts,
2. moving functionality to a newly created sibling class; and
3. insertion/removal of classes from a class hierarchy and redistribution of their functionalities.

5.1.2 Related Work

A large amount of research has been carried out on software maintenance and reverse engineering. The following works are relevant to this case study.

Wilde and Huitt (1992) discuss the difficulties that may be encountered in the maintenance of object-based systems. They note that the property of dynamic binding—e.g. through virtual classes in `C++`—provides a large amount of flexibility from the design point of view. However, it can create difficulties while tracing the dependencies of a call to a virtual method, because the actual method called may be one of a number of virtual implementations. Similarly, the use of polymorphism and inheritance may create an explosion of various dependencies of types class-to-class, class-to-method, method-to-method, method-to-variable etc. Object oriented practices encourage using a large number of small methods. Consequently, the code for a given task can be widely distributed, which makes the program difficult to understand. Furthermore, locating high level functionalities can be difficult, since a functionality is usually dispersed into various object classes. The authors also recommend analysis tools which should be provided by a maintenance framework to address the above issues.

Lientz et al. (1978) surveyed 120 organisations and analysed their *maintenance effort*, dividing the effort into the four categories given in 5.1.1. Their observation is that on average, 17.4 % of the maintenance effort is corrective, 18.2 % is adaptive, 60.3 % is perfective and 4.1 % is preventive.

Domsch and Schach (1999) present a case study in object-oriented maintenance in which the text-based user interface (UI) of a product which determined the number of power supplies required for a system configuration, was replaced by a graphical user interface (GUI). The additional requirements were: (i) the new software must send relevant graphical outputs to a printer, and (ii) the new product must run on 32-bit Microsoft Windows platforms. The requirements constrained the maintainer to use Microsoft specific APIs. The software engineer who developed the product was also the maintainer. The total maintenance effort was 116 man hours, 94.8 % of the which was perfective (development of the GUI), 3.2 % adaptive and 2 % corrective.

One of the important issues in re-engineering is the detection and location of design flaws which prevent efficient maintenance and further development of the system. Marinescu (2001) discusses a metric-based approach to detect such design flaws. The two most well-known design flaws are *god classes* and *data classes*. God classes tend to centralise the intelligence of the system, while data classes define data fields but have almost no methods except some accessor methods. Marinescu uses a metric-based approach for detecting god classes and data classes. A case study was done on an industrial project of 50,000 lines of `C++` code. The approach was highly effective, although there were design flaws, like duplicated code and the number of detected bugs in a class, which could not be addressed by his approach. The reason for this shortcoming is that these flaws cannot be detected by the chosen source code metrics.

In this chapter, the maintainability of the People Tracker and its evolution are discussed. An analysis is presented of the way it was re-engineered and the benefits that were obtained from the re-engineering. The findings have been published in Satpathy et al. (2002).

5.2 The People Tracker and its Maintainability

The system studied in this chapter is the People Tracking module of the `ADVISOR` integrated surveillance system introduced in Section 2.3.1.

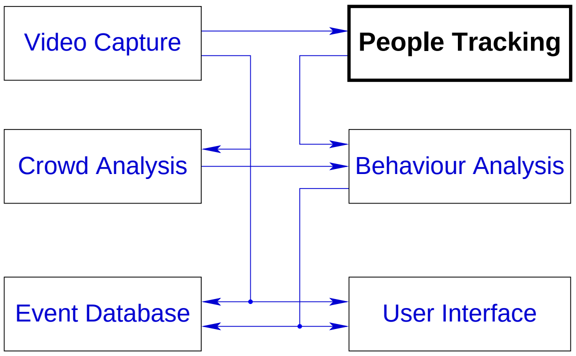


Figure 5.1: Data flow within the `ADVISOR` integrated system

5.2.1 Brief History

The original People Tracker was written by Adam Baumberg at the University of Leeds in 1993–1995 using `C++` running under `IRIX` on an `SGI`. It was a research and development system and a proof of concept for a PhD thesis (Baumberg, 1995). The main focus during development was on functionality and experimental features which represented the state-of-the-art in people tracking at that time. Only very limited resources were available. A simple process cycle was used for code development. The only documentation generated was a short programmer's manual (about 5 pages) describing how to write an application using the People Tracker software.

In 1995–1998 the code was used in a collaboration between the Universities of Leeds and Reading. The software was adapted so it could interoperate with a vehicle tracker which ran on a `Sun Microsystems/Solaris` platform (Remagnino et al., 1997). Only little functionality was changed and added during this time and no new documentation was created. Most of the programming on the People Tracker at that time was done by the original developer.

Starting in 2000, the People Tracker has been changed for its use within the `ADVISOR` integrated system. This new application required a number of major changes on different levels. The analysis presented here focuses on the changes carried out within `ADVISOR`, and especially on how the 1998 version of the People Tracker had to be adapted to make system integration possible.

Table 5.1 summarises the characteristics of the People Tracker at different stages of the project, starting in January 2000. These metrics were obtained using the *CCCC* (*`C` and `C++` Code Counter*) tool by Littlefair (2001). The size is given in Lines of Code (LOC), not counting empty or comment lines. The original version, as of January 2000, will be referred to as "PT_0".

Version	LOC	classes	of which instantiated	methods	global functions
PT_0 (Jan 2000)	19,380	154	71 (46.1 %)	1,690	271
PT_1 (Mar 2001)	24,797	191	80 (41.9 %)	1,913	286
PT_2 (Sep 2001)	25,306	183	97 (53.0 %)	1,714	25
PT_3 (Jun 2002)	16,472	122	82 (67.2 %)	1,231	9

Table 5.1: Measurements for different versions of the *Reading People Tracker*

5.2.2 Motivation for Re-design

The planned use of the People Tracker within the `ADVISOR` System introduced many requirements that could not be met by the original implementation. Most of the new requirements arose from the fact that the People Tracker would have to be closely integrated with other system components. Figure 5.1 shows how the People Tracking module is connected to the other components of the `ADVISOR` system. The use of the People Tracker within the `ADVISOR` system also meant moving it "from the lab to the real world" which necessitated many additional changes. The main *new requirements* for the People Tracker were as follows:

- The People Tracker has to be fully integrated within the `ADVISOR` system.
- It has to run multiple trackers, for video input from multiple cameras (original software: one tracker, one camera input).
- The `ADVISOR` system requires the People Tracker to operate in realtime on a standard off-the-shelf hardware PC to enable economical system integration.
- Within `ADVISOR`, the People Tracker has to run autonomously once it has been set up, without requiring input from an operator.

The status of the existing People Tracker was evaluated in relation to the new requirements. It was observed that the system had significant deficiencies which hindered the implementation of the required new functionality:

- The heavy use of global variables and functions meant that multiple cameras could not be used.
- Except a few pages of operating manual, no other documentation was available.
- Although the code was written in `C++`, it made a limited use of object oriented features like encapsulation and inheritance.
- There were very little comments in the source code, making the code difficult to read.

- The names of some of the classes and methods were misleading.
- The code was written for an `SGI` platform running under `IRIX`, making use of hardware- and software-specific functions.

It was therefore decided to re-design the People Tracker.

5.3 Approaches Taken for the Re-design

5.3.1 Stages of Work

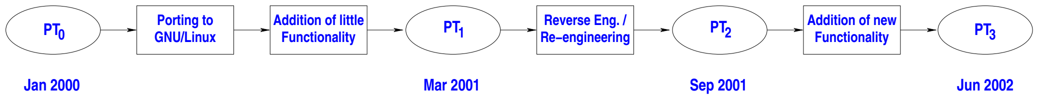

PT_0
Porting to GNU/Linux
Addition of little Functionality
PT_1
Reverse Eng. / Re–engineering
PT_2
Addition of new Functionality
PT_3
Jan 2000
Mar 2001
Sep 2001
Jun 2002

Figure 5.2: Sequence of work carried out during re-design

Figure 5.2 shows the sequence of steps which were followed to re-engineer the People Tracker. The original People Tracker is marked as “PT_0”, which was running on an `SGI` platform. In the first phase of the work the software was ported to a PC running `GNU/Linux`. The approach taken was “take code and compile”, replacing the `SGI`-specific functions (mostly for graphics) in the process. PT_0 contained calls to `SGI` video hardware and to a mathematical library which did not exist on the PC. These were replaced by more portable code.

Once the porting was complete, attempts were made to incorporate new functionality to the code. One example was a module to read XML files which adhered to a given *XML Schema* (XML Schema, 2001). This functionality was incorporated; during this time, the full extent of the software’s deficiency became evident. The decision was made to re-engineer the product before adding any new functionality. The product at this stage is referred to as “PT_1”. Its characteristics are shown in Table 5.1.

The aim of the re-engineering process was

- to understand the program and the design,
- to find and correct design flaws, and
- to generate all the software engineering artefacts—most importantly, documentation.

The class diagram was obtained by using the tool Rational Rose 2000e (2000). The analysis of the class diagram revealed that many class names did not reflect the inheritance hierarchy, a number of classes were found to be redundant, and many classes had duplicated functionality. The following correctional steps were performed:

- Redundant classes were removed.
- Global variables and functions were eliminated and their functionality distributed into both existing and new classes. Exceptions were global helper functions like `min()`, `max()` etc which were extracted and moved into one `C++` module.
- Many refactoring techniques (Demeyer et al., 2000) were applied, such as:
  - Filtering out functionality duplicated in similar classes and moving it into newly created base classes.
  - Re-distribution of functionality between classes and logical modules.
  - Re-distribution of functionality between methods.
- Meaningful names were given to classes and methods.
- Files were renamed to reflect the names of classes defined in them.
- PT_0 contained many class implementations in the header files; they were moved to the implementation (`.cc`) files.
- Assertions (Gries, 1981; Hoare, 1969) were introduced at strategic points in the existing code and in all of the new code, to aid in finding errors in the code at an early stage.
- From both *static analysis* and *dynamic analysis* (Richner and Ducasse, 1999), a requirement document and the UML artefacts like the Use Case, component, and package level sequence diagrams (Rumbaugh et al., 1999) were obtained. The UML diagrams are shown in Figures 5.5 through 5.7.

The product after the re-engineering step is referred to as "PT_2". Its characteristics can be seen in Table 5.1.

In the final step, the remaining part of the required new functionality was incorporated into the re-engineered product. This includes the addition of separate processing threads for each video input, addressing the synchronisation and timing requirements etc. A newly created master scheduler manages all processing, in order to guarantee realtime performance with multiple video inputs. Table 5.1 shows the characteristics of this version, which is referred to as "PT_3". PT_3 incorporates most of the functionality needed for its use within `ADVISOR` and improvements to the people tracking algorithms which make it appropriate for the application. The module has been validated against test data. Currently (June 2002), the final stage of system integration is being undertaken.

Stage	Porting	Little Func.	Re-Eng.	New Func.
Effort	8 MM	4 MM	8 MM	6 MM

Table 5.2: Distribution of the Maintenance Effort

5.3.2 Maintenance Effort

The maintenance effort[1] (in man months) in relation to the various stages in Figure 5.1 can be found in Table 5.2. Table 5.3 shows the percentage of maintenance effort by the four maintenance categories (Fenton and Pfleeger, 1996, p. 354–355). For comparison purposes, the industry average effort observed by Lientz et al. (1978) is also given. One can infer the following:

- Little corrective effort was necessary as the software was running without showing any significant errors at the start of the project.
- As the software needed to be ported to a different platform, the adaptive effort is higher than average.
- While the sum of perfective and preventive maintenance effort were found to be similar to the Lientz et al. (1978) average, there is a clear shift towards preventive maintenance. The main reason for this is that a lot of effort was put into improving the maintainability of the software, for instance by incorporating assertions into the code.

Category	Actual Effort	Industry Average
Corrective	8 %	17.4 %
Adaptive	31 %	18.2 %
Perfective	38 %	60.3 %
Preventive	23 %	4.1 %

Table 5.3: Maintenance effort by category, compared to the industry average

---

[1] Effort from Jan 2000 up to May 2002.

5.3.3 Code Size

Figure 5.3 shows the measures of code size as the software passed through the various stages shown in Figure 5.2. Initially, there was an increase in code size as new functionality was added to the system from version PT_0 to PT_1. The re-engineering phase further increased the code size because of the addition of extra documentation in the form of comments. Furthermore, some extra classes were generated to form additional layers between the main program and the classes implementing the people tracking functionality. These facts are observed by the stretch of curve from PT_1 to PT_2. Towards the end of the process there is a sharp decrease in code size because much unused functionality (*dead code*) which had been identified and marked as such during re-engineering was removed. Furthermore, many classes which were created for experimental purposes were removed. One such example is a class which adds noise to images to test the performance of algorithms in the presence of image noise. Since this functionality was not required in the final system, the class was removed.

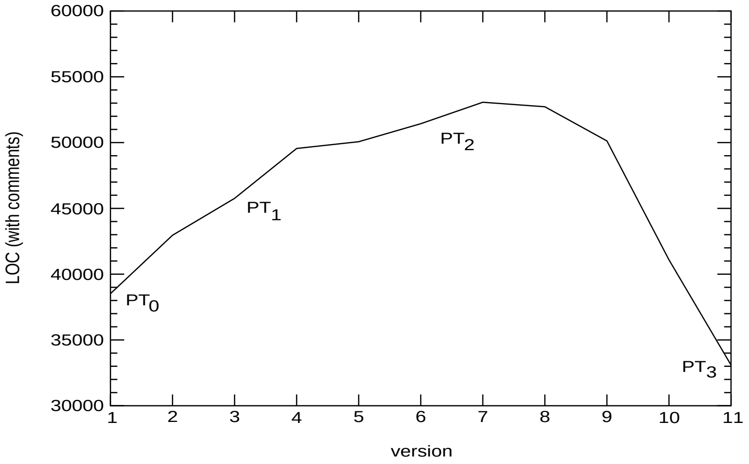


Figure 5.3: *Reading People Tracker*: Size in Lines of Code over Time

5.4 Further Analysis and Discussion

5.4.1 Generated Artefacts of the People Tracker

As a part of the re-engineering process, various software artefacts were generated from the code, and upgraded to reflect the latest version of the People Tracker. Most of the artefacts were manuals or UML diagrams. The initial class diagram which was obtained using the Rational Rose was the primary source of all activities. Both static and dynamic analyses were employed to obtain the remaining UML diagrams. They can be summarised as follows:

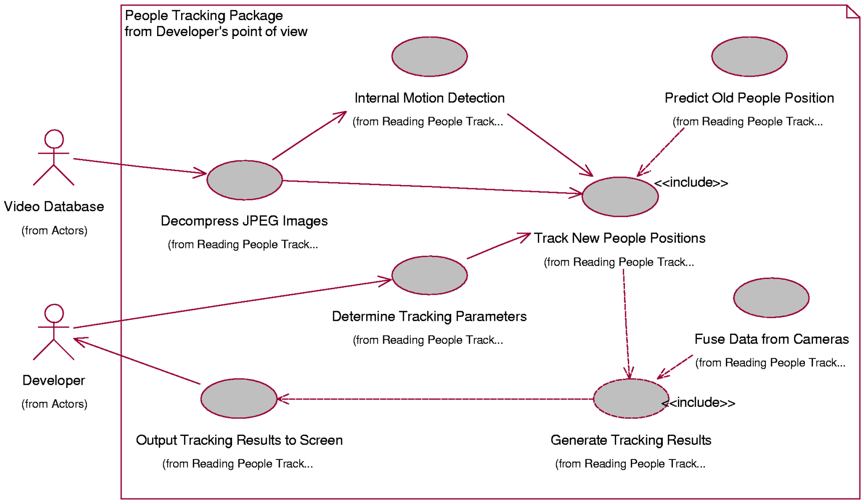

People Tracking Package
from Developer's point of view
Internal Motion Detection
(from Reading People Track...
Predict Old People Position
(from Reading People Track...
Video Database
(from Actors)
Decompress JPEG Images
(from Reading People Track...
<<include>>
Track New People Positions
(from Reading People Track...
Determine Tracking Parameters
(from Reading People Track...
Fuse Data from Cameras
(from Reading People Track...
Developer
(from Actors)
<<include>>
Output Tracking Results to Screen
(from Reading People Track...
Generate Tracking Results
(from Reading People Track...

Figure 5.4: Use Case for the People Tracker (in Standalone/Development Mode)

Use Case diagrams There are two Use Case diagrams: one when the People Tracker works standalone, and the other when the People Tracker works as a subsystem of the `ADVISOR` system. The diagrams are shown in Figures 5.4 and 5.5, respectively.

Class Diagram The initial class diagram was upgraded to incorporate the new class structure and the refactoring and other transformations. The diagram is not shown here because of its size.

Package Diagram Figure 5.6 shows the package diagram.

Sequence Diagram Figure 5.7 shows the high level sequence diagram of the People Tracker.

5.4.2 Experience with the Re-engineered software

The re-engineering of the People Tracker made the addition of new functionality much easier. The following examples illustrate this:

- All classes handling the data and tracking functionality for one camera (i.e. one video input) were encapsulated within one `Camera` class. As a result, the handling of multiple cameras simply amounted to creating multiple instances of the `Camera` classes. The original version (PT_0) of the software handled functionality such as the video input, tracking etc directly in the `main()` program, and only one video input was implemented.

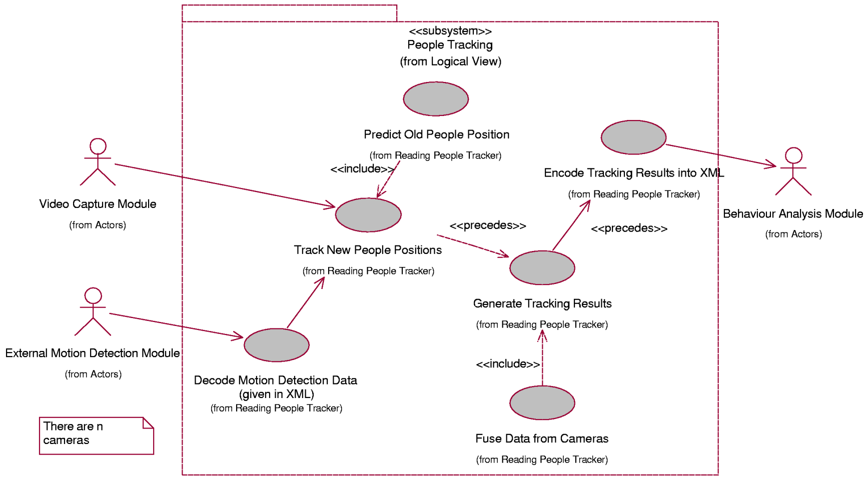

<<subsystem>>
People Tracking
(from Logical View)
Predict Old People Position
(from Reading People Tracker)
<<include>>
Encode Tracking Results into XML
(from Reading People Tracker)
Video Capture Module
(from Actors)
Behaviour Analysis Module
(from Actors)
Track New People Positions
(from Reading People Tracker)
<<precedes>>
<<precedes>>
Generate Tracking Results
(from Reading People Tracker)
External Motion Detection Module
(from Actors)
<<include>>
Decode Motion Detection Data
(given in XML)
(from Reading People Tracker)
Fuse Data from Cameras
(from Reading People Tracker)
There are n
cameras

Figure 5.5: Use Case for the People Tracker (as a Subsystem of `ADVISOR`)

- On a larger level, all `Camera` classes within one area of the underground station are contained within one `PeopleTracker` class. Hence the system design allows for scaling of the system simply by instantiating more than one `PeopleTracker` class. This was done keeping the aspect of scalability of the design in mind, although this functionality is not required at present.

- In the new design, all tracking results are contained within one `Results` class. As a consequence, the new requirement to write out the results in XML format was a very localised operation. Additionally, changing the design to reflect the new functionality was straightforward.

5.4.3 Improved Maintainability

The re-engineering of the software has improved its maintainability. This is shown here considering each of the maintainability categories separately.

Corrective Maintenance The software now uses assertions, hence it is easier to find bugs at an early stage. Also, the new design means that malfunctions of particular functionalities can now be easily localised and corrected.

Adaptive Maintenance The new software is no longer dependent on any particular hardware. All code now strictly follows the ISO `C/C++` standard, compliant with ISO/IEC 9899 (1999), as well as POSIX 1003.1c-1995 (1995) extensions for all multi-threading functionality. As a result, future porting of the People Tracker should be easy[2].

---

[2] In fact, this has been proven in April 2000 when the source code was compiled and linked under

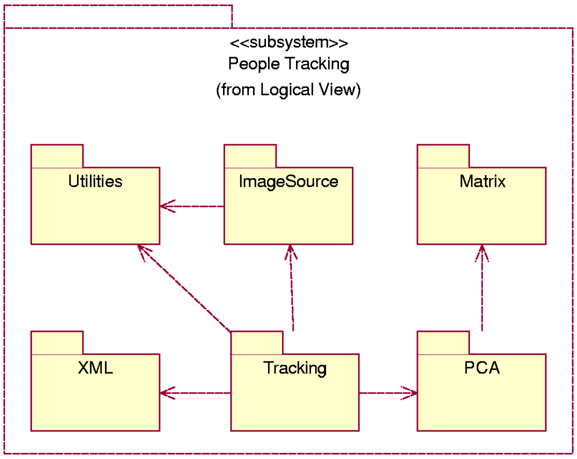


Figure 5.6: Software Packages of the People Tracker

Perfective Maintenance The software is now better documented and it has a clean structure. Furthermore, all the artefacts of the system are available and consistent with the source code. Hence, adding new functionality is easier, as already experienced, and discussed above.

Preventive Maintenance Assertions, which are now used, help to prevent malfunction of the code by making it easier to locate bugs. Additionally, the re-engineering and the new design have significantly improved the maintainability of the software.

5.4.4 Personnel Factors

The skills of the personnel involved in the work described in this chapter are as follows:

1. The author of this thesis, a PhD student having academic knowledge of software engineering and object-oriented programming but little experience in software maintenance. He was the team leader and carried out 60 % of the programming work himself.
2. A second PhD student having academic knowledge of software engineering, a moderate level of programming experience and no experience in software maintenance. He was mainly involved in the porting process and has completed 15 % of the overall work.

`Microsoft Windows 2000` within only 12 hours. This compares nicely to the 8 man months it took to port the software from `SGI` to `GNU/Linux`, as reported above, even if you consider that the second time around staff had had more experience with the software.

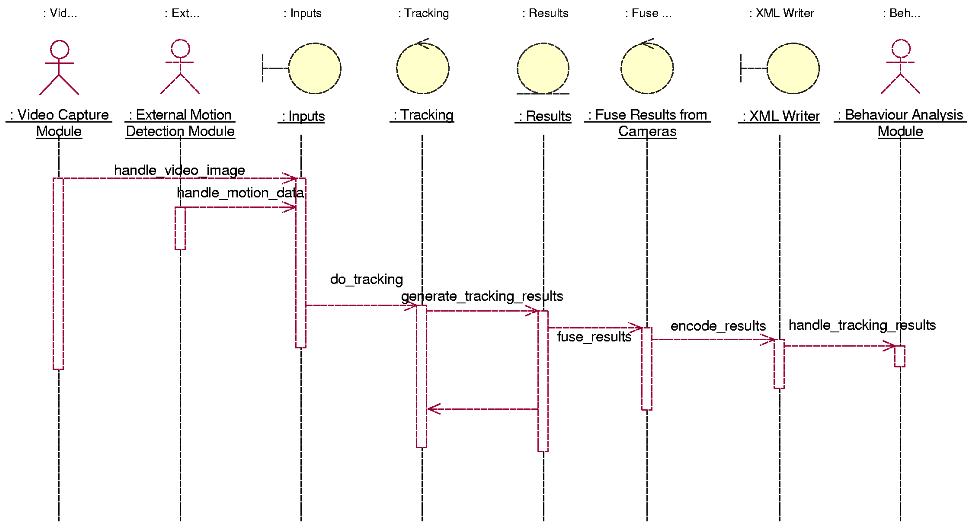

: Vid...
: Ext...
: Inputs
: Tracking
: Results
: Fuse ...
: XML Writer
: Beh...
: Video Capture Module
: External Motion Detection Module
: Inputs
: Tracking
: Results
: Fuse Results from Cameras
: XML Writer
: Behaviour Analysis Module
handle_video_image
handle_motion_data
do_tracking
generate_tracking_results
fuse_results
encode_results
handle_tracking_results

Figure 5.7: High level Sequence Diagram of the People Tracker within `ADVISOR`

3. Two undergraduate students having no knowledge of software engineering but good programming experience. They did 20 % of the task.

4. A senior researcher having no knowledge of software engineering and a moderate level of programming experience. He did 5 % of the work described here.

The lack of maintenance experience of the people involved was a prominent factor in making the software process inefficient. Instead of making a detailed initial work schedule, they started to implement new functionality at too early a stage. Only during this implementation process they concluded that re-engineering of the software was unavoidable in order to use it within `ADVISOR`. If re-engineering had been done at the beginning of the work, precious time could have been saved.

5.4.5 Lessons Learned

1. Any kind of maintenance activity must be preceded by adequate planning. In the present case, initially attempts were made to incorporate the new functionality and only when the attempts failed was it decided to re-engineer the software.

2. Some training should be given to maintainers so that they can recognise unmaintainable code. In the present case, the maintainers took significant time to learn about the extent to which the code was not maintainable. At this stage, some tool support could aid in determining the state of the code. The tool

could use indicators like the number of global variables and global functions, cohesion and coupling, amount of documentation etc.

3. Initially, the maintainers felt a strong resistance against re-engineering the software. Even when they found out that adding new functionality would be difficult with the current state of the code, they hesitated to re-engineer the software because it was not a part of their given task. This seems to be a common problem in software projects: making the best tradeoff between the short-term benefits of a quick addition of new functionality and the long-term benefits of re-engineering to obtain a product which is better maintainable. While developers are more likely to determine the necessity to re-engineer a product, managers tend to resist it and focus on short-term benefits. Re-engineering takes up a lot of time and resources in the short term, although in the long term it might save time due to the increased maintainability.

4. A precise and good process model of the re-engineering process should be in place before beginning the re-engineering. In the present case, the trial-and-error approach consumed a lot of precious time.

5. A correct design should be a pre-requisite to implementing new functionality. Design documents and other artefacts should always remain consistent with the source code.

6. In the present case, refactoring transformations were applied after locating problem spots manually. At this stage, tool support should be used to recognise bad classes (for instance, *god classes* and *data classes*, see Marinescu, 2001).

7. Developers should stick to standard languages and avoid language extensions. Similarly, the product should not depend on specific hardware to run. Where software- or hardware-specific functions are required, e.g. for optimisation, they should be isolated and adequately documented so as to ease future porting operations.

8. It was observed that a different level of domain knowledge was needed for the different maintenance activities. Domain knowledge was not a pre-requisite for performing the porting task. As far as re-engineering was concerned, very little domain knowledge was required for performing the static analysis, whereas dynamic analysis required a significant amount of domain knowledge. A moderate level of domain knowledge was necessary when adding new domain specific functionality.

9. The maintainers of the People Tracker had occasional short discussions (over lunch table or during coffee break) with experienced software engineers in the

University. This guidance increased their awareness of the technical aspects of software maintenance. Consultation with an experienced software engineer is therefore highly recommended before and during the maintenance task.

5.5 Conclusions

In this chapter, the People Tracker module of the integrated system `ADVISOR` has been the object of a case study. The medium sized People Tracker software was originally developed in a University environment. The software process yielded a code with very low level of maintainability. Therefore the software was re-engineered so that it could be used in the "real world" as a subsystem of `ADVISOR`. The team leader of the maintenance operation is the author of this thesis. None of the people involved in the re-engineering task had any maintenance experience and therefore the whole process was carried out in a suboptimal manner. The people involved took occasional help from experienced software engineers who provided important guidance when it came to software engineering aspects of the work,

The maintainers successfully re-engineered the product and upgraded it with new functionality. The product has been found to be satisfactory by the project partners and it is now (June 2002) in the final stage of its integration with the other subsystems of `ADVISOR`. Furthermore, the maintainers are satisfied with the work they have done. The analysis presented here shows that the re-engineering process has been effective in achieving a high level of maintainability.

The following can be concluded from the observations:

- Tool support or expert advice could aid in determining whether software should be re-engineered before any other maintenance task is carried out.
- Increasing the maintainability of software by re-engineering techniques can make the addition of new functionality and porting more efficient, thereby saving time to accomplish a maintenance task.
- Surprisingly little domain knowledge was necessary for porting or re-engineering the software. Domain knowledge was needed to extract dynamic information from the code, and partly to add new functionality. The availability of design documents can to a certain extent make up for missing domain knowledge.

Chapter 6

Process Diversity and its Implications for the People Tracker

When dealing with a piece of software the size and age of the Reading People Tracker one invariably needs to consider the processes used to build and maintain it. In this chapter the evolution of the People Tracker is analysed with respect to the diversity of these processes. The analysis shows the influence which different contexts, applications and techniques had on the quality of the processes and on the software itself. A new classification scheme of process diversity into *latitudinal* and *longitudinal process diversity* is presented which helps to understand the consequences of process diversity.

6.1 Introduction

It is now widely accepted that most software evolves significantly over its lifetime. One of the implications of this discovery is that there are relationships between software processes and software evolution—that is, the way in which software is developed and maintained may have longterm effects on the software itself. In this chapter, a particular aspect of this issue is examined, namely the effects of *diversity* in these processes on the quality of both software products and processes. The main contribution is to distinguish between two broad kinds of process diversity; *latitudinal* and *longitudinal process diversity*. This classification provides a conceptual framework for a better understanding of process diversity and thereby helps managers to apply specifically tailored countermeasures to control the negative effects of a particular type of diversity. These concepts are illustrated in the case of the People Tracker, and ways are suggested in which they appear to affect both product and process quality.

6.1.1 Concepts and Management of Process Diversity

6.1. Definition (PROCESS)**:** *A process is a collection of activities carried out by people and/or machines that are intended to achieve some desired outcome, are related to each other in time and have inputs and outputs.*

This mostly corresponds to the *development processes* and some of the *supporting processes* as defined by ISO/IEC 12207 (1995). For management purposes, processes are often recursively decomposed into *subprocesses* reach what are called *atomic processes*. The possible granularity of processes covers a very wide range, from high-level business processes to finely detailed processes for software maintenance. In order to understand process diversity, one needs to situate software processes in the context of larger business processes.

A process is represented by a *process model* and executed within a *process environment* Doppke et al. (1998). The process environment links together the people executing the process (e.g. managers and software developers) and any domain-specific tools that may be used. As processes do not have unique representations, and they can be executed within differing process environments, processes invariably differ. In this context, *process diversity* is considered as it occurs when a project is executed within a different process environment, its impact is studied. This diversity can happen either concurrently (e.g. in multi-team projects) or when a project encounters different process environments during its life-cycle. Process diversity can have negative consequences. Two well-known examples are problems during system integration (as project partners might follow differing software processes), and problems arising from software re-use in a new environment.

Diversity in software processes is usually inevitable and managing it can be a delicate balancing act. Too much process diversity can lead to chaos but too little may suppress creativity and thus lead to missed opportunities. Whilst day-to-day process management will be more concerned with deliverables, schedules and budgets, the management of process improvement should periodically review whether a satisfactory balance between uniformity and diversity in software and other processes is being achieved. This work supports this aspect of process review and improvement by proposing a simple high-level model of process diversity that distinguishes two broad categories which have different sources and impacts.

6.1.2 Related Work

The work described in this chapter is related to two established themes in software engineering research, namely process improvement and software evolution.

The need for quality improvement and cost reduction in software production and maintenance has led to a research emphasis on *process improvement*. Carnegie Mellon University's Software Engineering Institute (SEI) has set up a dedicated group of Software Process Improvement Networks (*SPIN*s, see `http://www.sei.cmu.edu/collaborating/spins/`) with the aim of connecting individuals involved in improving software engineering practice. Research and case studies show how much money an improved software process can save a company (Dion, 1993) but also stress the complexities involved in process improvement (Cattaneo et al., 1995; Herbsleb et al., 1994; Humphrey, 1992; Sharp et al., 1999). A number of process models and standards have been developed to analyse and improve software processes, most prominently the *Capability Maturity Model* (*CMM*), but also ISO standards such as ISO 9000 and ISO 15504 (see Paulk (2001) for a review of these and other standards). Process improvement is usually carried out using a process model, and a measurement framework like the *Goal/Question/Metric method* (*G/Q/M*) (van Solingen and Berghout, 1999).

The identification of *software evolution* as a research topic originated in Lehman's pioneering studies of the long-term development of IBM systems in the 1960s (Lehman, 1969) which resulted in the formulation of Lehman's widely respected "Laws of Software Evolution" and his *S-P-E* classification of information systems (Lehman, 1980; Lehman et al., 1997) which will be addressed in more detail in Section 6.2.3. Since then, the field has gradually broadened as software engineering researchers have studied relationships between software evolution and other topics, and deepened as various aspects have been studied in detail. Nevertheless, many of the core concepts and approaches that are currently used can be traced back to Lehman's work.

The research presented in this chapter has been published in Siebel et al. (2003).

6.2 Process Diversity

Diversity in software processes can occur on many different scales, and its impact on product and process quality can also cover a wide range. In order to understand the likely impact of particular examples of diversity, it is helpful to distinguish two high-level categories:

Latitudinal process diversity: This category describes the kind of variation that occurs when diverse processes operate concurrently within the same project. This is often observed in multi-team projects (Herbsleb and Grinter, 1999).

Longitudinal process diversity: This is the variation that occurs in software processes over time. An example which is often observed is the transition from development to maintenance phases in a project.

Thus latitudinal and longitudinal process diversity are not properties of individual processes; they are properties of the system (in this case, a software development or maintenance system) that provides the context for processes—the *process environment*. Consequently, these properties may be of greatest interest to people who are responsible for managing, designing or improving processes, rather than directly executing them.

These high level categories of process diversity need to be distinguished from finer-grained categories. In particular, it is important to distinguish between process diversity as an emergent property of a system for developing or maintaining software (which is the subject of this work), and the intentional introduction of diversity as a software engineering technique—as in, for example, *N-Version Programming* (Avizienis, 1985).

6.2.1 Latitudinal Process Diversity

The term *latitudinal process diversity* refers to the phenomenon of diverse software processes operating concurrently. This type of process diversity is often found in projects that span company boundaries. It has its most severe impacts when software from several project partners is integrated together (Herbsleb and Grinter, 1999). This section proposes a model for understanding its main dimensions. The model is intended to describe diversity in a neutral way, but the underlying assumption is that some latitudinal process diversity can be creative (or at least harmless) but too much diversity can be harmful (or at least distracting). This may be too simplistic; it may be the case that, for example, the approach to managing diversity is as important as the extent of diversity but one cannot draw sound conclusions about this from a single case study. This is a research area where software engineering should perhaps seek more insights from disciplines such as occupational psychology, social anthropology and management where issues of work group formation, culture and interaction have been studied.

Role Differentiation

In the simplest kind of software project, the roles of customer, developer and user are subsumed in a single person. However, as projects increase in complexity, these roles tend to differentiate in two ways:

1. The various functional roles (customer, developer, maintainer etc.) become more distinct and behave as separate stakeholders in the system, each one having characteristic objectives, concerns and priorities.

   The recent IEEE Standard on software architecture descriptions, IEEE Std-1471-2000 (2000), reflects this idea by stating that any viewpoint of a software system is addressed to one particular stakeholder of the system. Therefore any viewpoint may cover only a small subset of the concerns identified in the architectural description of the system.

2. Particular roles become shared by individuals, teams or organisations, who may have different notions about how the role should be carried out.

Both kinds of differentiation can lead to latitudinal process diversity. At any moment, the various units (individuals, teams, departments) within an organisation may be involved in:

- performing different processes (because the units occupy different roles, e.g. maintainer, product-line architect, customer); and/or,
- performing the same process in different ways (because the units have different cultural or professional approaches to their roles, e.g. programmers and technical writers producing system documentation).

The findings by Bianchi et al. (2001) show that increasing the number of staff members increases the risk of defects and hence rework. This, in turn, generates a greater gap between expected and real staff requirements. The study also confirms that in distributed processes there is a greater need for communication among the working members than in colocated processes.

Cultural Diversity

As an engineering product, software is affected by the environment in which it is produced. A variety of sociological factors (e.g. the socio-cultural backgrounds of team members, the structure and business practices of organisations) may lead to differences in the processes teams use to produce software—see Carmel (1999) for examples. Thus cultural diversity of various kinds can lead to latitudinal diversity in software processes. This becomes most apparent in multi-team projects where pieces of software from different teams need to be integrated into one system.

6.2.2 Longitudinal Process Diversity

The relationship between process diversity and software evolution can be seen most clearly in the longitudinal case. A fundamental kind of longitudinal process diversity occurs when software is required to evolve. For example, a program may be originally developed as a proof-of-concept for some abstract computation with essentially static requirements. Subsequently, the program may be integrated into an information system that supports an evolving business process. This brings the program within the influences of a more dynamically evolving system, and its software process may need to be adapted accordingly.

The concept of longitudinal process diversity implies a long-term viewpoint on software processes. In this view, the detailed definitions of individual steps in a software process are less important than the overall scope and configuration of the process considered as a system. Over the long term, factors such as the sources of change to the software product and the nature of the feedback paths in the process become more influential; these are also important factors in software evolution (Chatters et al., 2000). Conversely, factors such as the choice of programming language or the use of specific software engineering techniques, like design reviews or code inspections, become less influential. Thus a longitudinal process diversity viewpoint has strong similarities with, and complements, a software evolution viewpoint. They share many concerns (e.g. system maintainability) and modelling techniques (e.g. system dynamics) but differ in whether the focus is on the process or its product.

6.2.3 Evolution-oriented Models of Software Processes

The relationship between longitudinal process diversity and software evolution will be illustrated using two software process stereotypes that are distinguished by their propensity for software evolution. *Stereotype 1* models a process that is appropriate for software with a low propensity for evolution, *Stereotype 2* one for software with a high propensity for evolution. These two represent the extremes of the range that is likely to be found in practice. Longitudinal process diversity can be understood as movement within this range during the lifetime of a software product.

Both stereotypes use the systemic viewpoint described in Section 6.2.2 to situate the engineering process of maintaining software in a broader context. Consequently, most of the details (and diversity) of software engineering techniques have been subsumed into a single node *Generic Software Change Process*. This is a generic placeholder; it represents a software change process in relation to any life-cycle model, e.g. the *Waterfall model* or *agile processes* like *XP* (Beck, 1999). The main features of these process models are illustrated by schematic process diagrams in which the boxes represent (sub)processes and the arrows represent information flow between them. The starting point for all maintenance activities which originate from outside sources have also been combined into a single node *Assess Impact of Exogenous Changes*. Exogenous changes include regulatory, technological and market-driven changes. The diagrams are intended to highlight the distinguishing characteristics of each model and not to be complete specifications of software processes.

Stereotype 1: Low Propensity for Software Evolution

The process model shown in Figure 6.1 is intended for software products with a low propensity for evolution. Their functional requirements are likely to evolve slowly, if at all. In terms of Lehman's *S-P-E* taxonomy, they fall into the *S-type* (*Specified*) and

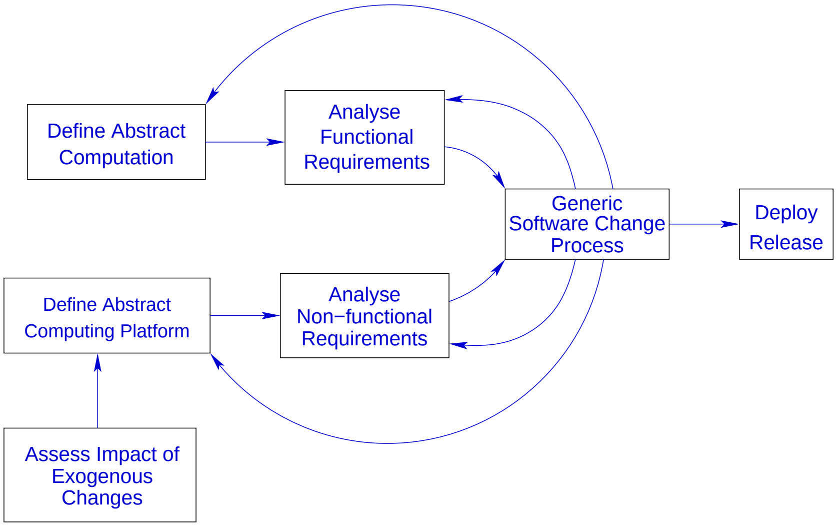

Define Abstract Computation
Analyse Functional Requirements
Generic Software Change Process
Deploy Release
Define Abstract Computing Platform
Analyse Non−functional Requirements
Assess Impact of Exogenous Changes

Figure 6.1: Software Process Model, Stereotype 1—Low Propensity for Evolution

P-type (*Problem-solving*) categories. Implementations of *P-type* software may need to be adapted occasionally to take account of changes in their technical environments. In the case of an *S-type* program, by definition, its requirements provide a complete description of the problem to be solved and its implementation does not require any design compromises. Consequently, there are no strong pressures for such programs to evolve. For a *P-type* program, the first condition is slightly relaxed; its functional requirements may be an abstraction (which can be redefined) from the problem to be solved, but the problem itself is static. A *P-type* program has significant non-functional requirements that must be reconciled with the functional requirements by the program's design. Changes in the technical environment can require the program to be adapted. For example, if a manufacturer stops supporting particular hardware or compilers, a program which depends on them may have to be ported to a new platform if it is not to become unusable. Conversely, when new technical capabilities become available, they can trigger a reconsideration of design compromises, and possibly a redefinition of the problem abstraction (but not the problem itself) that is represented by the functional requirements.

Stereotype 2: High Propensity for Software Evolution

When software is embedded in an information system that supports a business (or social) process, its software process inevitably becomes more complex than the model in Stereotype 1. Parnas (1994) has used the notion of *software aging* to characterise the additional pressures that affect software and software processes in this situation.

One approach to understanding these differences in process can be found in

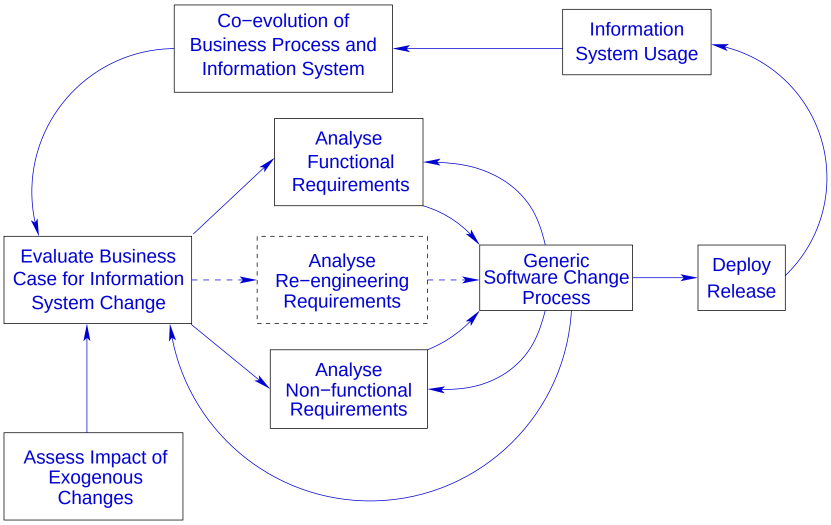

Co−evolution of Business Process and Information System
Information System Usage
Analyse Functional Requirements
Evaluate Business Case for Information System Change
Analyse Re−engineering Requirements
Generic Software Change Process
Deploy Release
Analyse Non−functional Requirements
Assess Impact of Exogenous Changes

Figure 6.2: Software Process Model, Stereotype 2—High Propensity for Evolution

Lehman's "Laws of Software Evolution" (Lehman et al., 1997) and the studies that he conducted within the *Feedback, Evolution And Software Technology* (*FEAST*) projects (e.g. Chatters et al., 2000). Figure 6.2 shows Stereotype 2, which is a model of the software process that elaborates Stereotype 1 to take account of the findings from Lehman's work that are most relevant to longitudinal process diversity. The software process modelled in Stereotype 2 is intended for programs which fit into Lehman's *E-type* (*Embedded* or *Evolving*) category (Lehman, 1980). The increase in complexity in this model compared to Stereotype 1 arises from three principal characteristics of *E-type* programs:

1. Both initial development and subsequent changes to the software are affected by an evolving business case which determines both the functional requirements and constrains the choice of computing platform. The business case can change at any time for reasons that are ultimately open-ended. Consequently, it is no longer relevant to distinguish between the parts of the model that are or are not subject to evolution.

2. The usage of the system produces business benefits. If the system is successful, this will tend to stimulate co-evolution (Warboys et al., 2000) of the information system and the business process. This will often result in unanticipated changes in the system's requirements via a revised business case.

   For example, a successful stock control system may create opportunities for, say, just-in-time ordering or the development of an "extranet" with trusted trading partners.

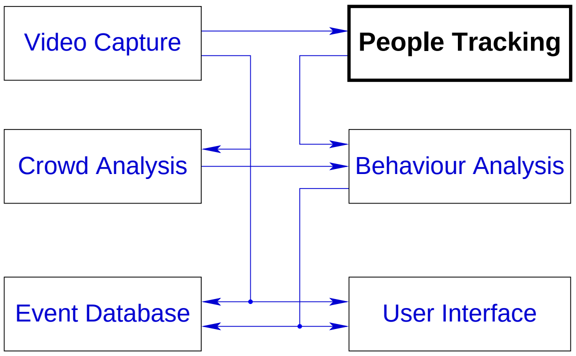


Figure 6.3: People Tracking as one of six subsystems of `ADVISOR`

3. One of the side-effects of the increased propensity for evolution is that periodically it may be necessary for the business case to include some re-engineering of the information system (Waters and Chikovsky, 1994; Weide et al., 1995). This path through the process model is shown with dashed lines and boxes in Figure 6.2. Periodic re-engineering becomes necessary because the co-evolution of the business process and the information system tends to result in continual additions of functionality to the system. Consequently, the software tends to grow both in size and complexity unless work is done to counteract this growth. The increased size and complexity of the software tends to reduce the productivity of the maintenance effort. If the re-engineering work is postponed for too long, there will be an increased risk that maintenance productivity would decline to a level where the business case cannot be satisfied any longer.

Latitudinal process diversity is less closely related to software evolution than the longitudinal kind is. Nevertheless, there is an implicit connection between these concepts. The efficient operation of the processes modelled in Stereotypes 1 and 2 depends in part on the quality of communication between the human participants in the process. It is a plausible conjecture that high levels of latitudinal process diversity will tend to undermine the quality of these interactions.

The following sections discuss the Reading People Tracker as an example case which exhibits both latitudinal and longitudinal process diversity, and where the corresponding difficulties can be clearly observed.

6.3 Description of the People Tracker Software

The software package studied here is the People Tracker as a subsystem of the integrated surveillance system `ADVISOR` (Figure 6.3). `ADVISOR` is being built as part of

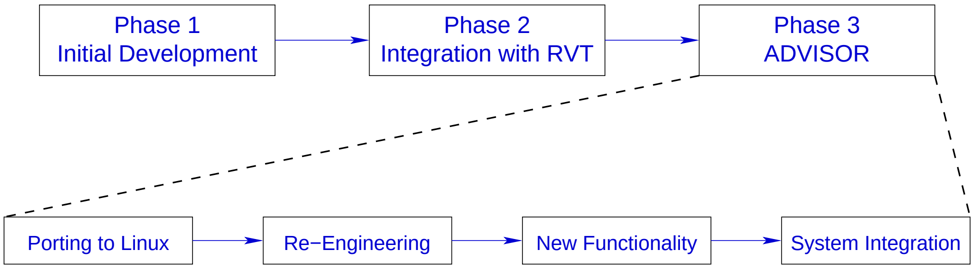


Figure 6.4: Main development and maintenance phases of the Reading People Tracker

a European research project involving three academic and three industrial partners. The task of the People Tracking subsystem is to automatically analyse images from one or more camera inputs. A stream of video images is continuously fed to the system, and the subsystem has to detect all the people in the image and track them in realtime.

6.3.1 Brief History

The evolution of the People Tracker can be divided into three main phases, extending over 10 years. Figure 6.4 shows an overview of these phases.

Phase 1: Initial Development (1993–1995)

The original People Tracker was written at the University of Leeds in 1993–1995 using `C++` running under `IRIX` on an `SGI`. It was a research and development system and a proof of concept for a PhD thesis (Baumberg, 1995). The main focus during development was on functionality and experimental features which represented the state-of-the-art in people tracking. Only very limited resources were available for the design and maintenance of image processing libraries developed and used within project. For code development a simple process cycle was used to develop the software. The only documentation generated was a short programmer's manual (about 5 pages) describing how to write an application using the People Tracker software.

Phase 2: Integration with a Vehicle Tracker (1995–1998)

In 1995–1998 the code was used in a collaboration between the Universities of Leeds and Reading. The software was changed to interoperate with the *Reading Vehicle Tracker* (*RVT*) which ran on a `Sun Microsystems/Solaris` platform (Remagnino et al., 1997). Little functionality was changed or added during this time and no new documentation was created. Most of the functional changes made to the People Tracker during this phase were carried out by its original developer.

Phase 3: Re-design and Integration into the `ADVISOR` System (2000–2002)

Since 2000, the People Tracker has been adapted for use within the `ADVISOR` system shown in Figure 6.3. As discussed in Chapter 5, its planned use within `ADVISOR` carried many requirements the original implementation could not meet. Hence, after porting it from `SGI` to a `GNU/Linux` PC, it was decided to re-engineer the software. After the re-engineering step new functionality was incorporated into the software and missing software artefacts were generated. Among these is a comprehensive manual (Bouffant et al., 2002) which documents the new software process which is now to be followed for all maintenance work. As the last step within Phase 3, the People Tracker was integrated into `ADVISOR`.

6.3.2 Current Status and Outlook

An important result of the re-engineering and documentation process is a new software process which is now in place. This documented software process defines

- detailed coding standards, including design patterns (Fowler et al., 1999)
- configuration management (includes the use of `CVS` for concurrent version management)
- a number of “HowTos” which document how to implement new classes for the most important functionalities (e.g. acquiring images, tracking).

One can compare the software process followed in Phase 1 with the one defined by *CMM* (Paulk et al., 1993) in “Level 1” while some *Key Process Areas* are now comparable to those in *CMM* “Level 3”. The current (June 2002) status of the People Tracker can be summarised as follows:

- The software is fully operational with all necessary functionality implemented.
- It can run either in standalone mode or integrated within `ADVISOR`.
- The re-engineered People Tracker exhibits a high level of maintainability and all software artefacts are consistent with the implementation.
- The code is being maintained under `GNU/Linux`. For integration purposes within the `ADVISOR` project, releases of the People Tracker subsystem have been successfully compiled and integrated under `Microsoft Windows 2000`.

The testing phase for the Tracker running in standalone mode has been successfully completed. The testing phase for its use in the integrated system is in progress. Within a few months from now, a prototype of the `ADVISOR` system will be tested in

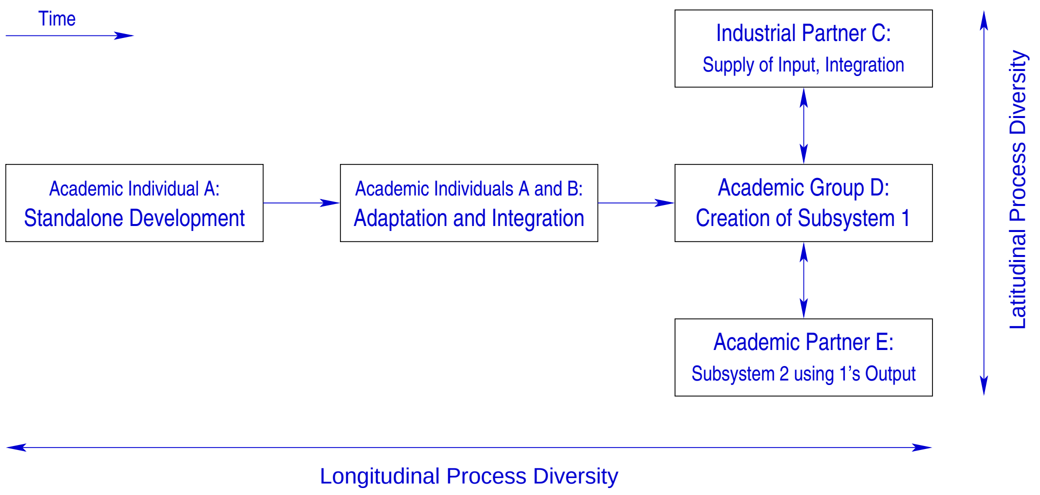

Time
Industrial Partner C:
Supply of Input, Integration
Academic Individual A:
Standalone Development
Academic Individuals A and B:
Adaptation and Integration
Academic Group D:
Creation of Subsystem 1
Academic Partner E:
Subsystem 2 using 1's Output
Latitudinal Process Diversity
Longitudinal Process Diversity

Figure 6.5: Latitudinal and Longitudinal Process Diversity

underground stations in Brussels and Barcelona. There are a number of proposals for extending the use of the tracking system. An example is to use it to monitor the activity of a species of rare bat (Mitchell-Jones, 2002).

6.4 Analysis of the Process Diversity for the People Tracker

Over the 10 years that the People Tracker software has been maintained and used, a wide range of software processes has been employed. In the following sections, this phenomenon is examined in more detail, differentiating between the *latitudinal process diversity* experienced during Phase 3 and the *longitudinal process diversity* occurring over all three phases. The relationships between the phases of the People Tracker's history and its latitudinal and longitudinal process diversity are summarised in Figure 6.5. An analysis shows whether and how the impact of process diversity on the project differs with its type, latitudinal or longitudinal.

6.4.1 Latitudinal Process Diversity: Diversity between Co-operating Groups

The use of the People Tracker within the `ADVISOR` project led to increased latitudinal process diversity in Phase 3, compared with Phases 1 and 2. The principal change was that the academic maintainers, denoted by D in Figure 6.5, had to co-operate with the industrial partner C, and also with an academic partner E from a different country. This meant more actors with distinct roles, as well as different (kinds of) organisations sharing roles, as discussed in Section 6.2.1. The following description will concentrate on these three out of the six groups within `ADVISOR` as the others do not directly interface with the People Tracking subsystem (cf. Figure 6.3).

In the collaboration with the industrial partner C, the maintainers of the People Tracker and their partners experienced many "cultural" differences. For example, there were differences in the partners' approaches to the use of standards and choices of tools and platforms. The new People Tracker subsystem was maintained under `GNU/Linux` on a PC platform. Academic group D adhered to the ISO `C/C++` standard, compliant with ISO/IEC 9899 (1999), to maximise portability. Towards the end of the development, all the design and the implementation were consistent with each other. The documentation, a programmer's manual and a user's manual, were typeset using LaTeX. The module developed by industrial partner C, on the other hand, was developed under `Microsoft Windows 2000` using `Microsoft Visual C++`. There was no plan to use their modules on different platforms within this project, and hence their implementation makes use of `Microsoft Windows`-specific functions, rendering it not easily portable. Their interface and other documents were written using `Microsoft Word` which could not easily be read by academic group D.

A consequence of these differences occurred when the People Tracker code, maintained under `GNU/Linux`, had to be compiled with `Microsoft Visual C++`. It became clear that `Microsoft Visual C++` could not compile the ISO C code as its compiler is not compatible with that standard. The resolution of this incompatibility required resources that neither partner had budgeted for.

As for the roles of involved people, industrial partner C was not only responsible for the system integration but were also the project leaders. This creates a situation of un-even partners, something which is unusual for academic groups and hence required adaptation on both sides.

6.4.2 Longitudinal Process Diversity and Transition between Process Models

As the People Tracker evolved through the three phases described above, its process model changed accordingly. During Phase 1 the process was similar to *Stereotype 1* while during Phase 3 it more closely resembled *Stereotype 2*.

During *Phase 1*, the main design and programming work was carried out by an academic individual. The goal was to develop a people tracking application which would exceed existing applications in performance. The People Tracker uses a large image processing library which forms an essential part of the software. This library had to be written in parallel to the main code development. During the process cycle testing and development were closely connected, and both were in the hands of a single person. Therefore there were only very limited resources available to develop and maintain the image processing library. Within other projects in the same research group image processing algorithms had already been implemented. However, no code was shared between projects which meant that these algorithms had to be implemented from scratch. Being an academic research project, there were also few fixed deadlines for the delivery of code. This meant that all parts of the code could be freely changed at any given time, and many modules were actually changed or re-written many times before the final delivery. Once the software and the PhD thesis were delivered, no functional changes needed to be carried out.

Phase 2 brought only a slight change in the software process. The system was adapted to incorporate a few new requirements, and the interaction with the vehicle tracker module was very loose—the two programs were not even compiled together. The original developer of the software from Phase 1 was involved and carried out many of the functional changes. This reduced the necessity for a change of processes from Phase 1 to Phase 2 as the necessary knowledge about the system was available without the creation of documentation etc.

None of the people involved in Phases 1 and 2 participated in *Phase 3*. This resulted in Phase 3 being completely de-coupled from Phases 1 and 2, the only linking element over time being the source code. Since there was practically no documentation available for the software, the design methodologies employed during the process were necessarily different from the ones employed before.

The following new requirements necessitated the creation of a process cycle which is close to Stereotype 2 (compare Figure 6.2):

- close integration with another `ADVISOR` module (compiled together) required interaction with project partners; interfaces needed to be agreed and delivery deadlines had to be kept.
- the significant functional changes necessitated re-engineering analysis and changes through refactoring and re-design. For example, one new requirement was that more than one camera could be used. The use of global variables in the Phase 2 code made this impossible.
- the People Tracker subsystem had to be de-coupled from software- and hardware-dependent functions, like `SGI` hardware graphics routines, so that it could be maintained under `GNU/Linux` but used (within `ADVISOR`) under `Microsoft Windows`.
- a strict project schedule with deliverables and milestones was introduced. Based on deliverables and `ADVISOR` steering committee decisions, software requirements were changed which meant that also a further feedback loop was in place.

Version	LOC	classes	of which instantiated	methods	global functions
pre-Phase 2	20,454	154	71 (46.1 %)	1,695	271
post-Phase 2	19,380	154	71 (46.1 %)	1,690	271
Phase 3, a	24,797	191	80 (41.9 %)	1,913	286
Phase 3, b	25,306	183	97 (53.0 %)	1,714	25
Phase 3, c	23,581	178	85 (47.8 %)	1,664	24
end Phase 3	16,472	122	82 (67.2 %)	1,231	9

Table 6.1: Measurements of the People Tracker over Time

- the planned use of the People Tracker for related, but different applications (like tracking bats, see Mitchell-Jones, 2002) or scaling of the system for larger applications inspired the generalisation of tracking concepts within the software, keeping in mind possible future uses.

6.4.3 Metrics and Further Analysis

Table 6.1 shows a brief summary of the characteristics of the People Tracker at different stages of its lifetime. The metrics were obtained using the *CCCC* (`C` *and* `C++` *Code Counter*) tool by Littlefair (2001), which is based on the metrics proposed by Chidamber and Kemerer (1994). The size is given in Lines of Code (LOC), not counting empty or comment lines.

Code Size The code size as well as the number of classes and methods has varied considerably. Up to the first stage within Phase 3, functionality was simply added without removing much unused functionality. Nevertheless, very little code was added during Phase 2 when the People Tracker was changed to interoperate with the Reading Vehicle Tracker.

When the re-engineering stage within Phase 3 started, there was also little change in code size. The explanation the developers gave is that at this stage, unused functionality was identified and marked as such. However, until the end of Phase 3, while new functionality for `ADVISOR` was added, most of the unused legacy functionality was kept in case it would be needed again. When it was finally removed in the last version examined here, one can see the sharp decrease in code size.

Global Functions The original code contained a large number of global functions. This number was substantially reduced during the re-engineering stage and again when most of the unused functionality was removed.

From our experience of Phase 3, one can infer several connections between the process changes described earlier and the product characteristics described above:

- The new software process, being closer to *Stereotype 2*, includes a new design strategy, which observes more dependencies and also considers possible future uses of the software. For example, one of its effects is the reduction of global functions to a minimum.

- One result of the re-engineering step in Phase 3 is that the size of the code was reduced and the proportion of used functionality increased. This can be seen in the percentage of instantiated classes given in Table 6.1. While in the original version of the software only 46.1 % of the defined classes were instantiated, this percentage has now gone up to 67.2 %. The reduction in redundancy directly improves software maintainability.

- The generated software artefacts and the re-designed code structure make it easier to understand and change the software. The current maintainers have already noticed the change as they compared the way in which students approached the unknown code when they started to work on the project, before and after the re-engineering process.

6.5 Lessons Learned

The proposed distinction between latitudinal and longitudinal process diversity is intended to help software managers analyse process diversity by providing a conceptual framework. Using this framework, corrective or preventive measures can be related to a particular type of process diversity. In this sense, our classification modularises process diversity.

In process assessment and improvement, e.g. using G/Q/M (van Solingen and Berghout, 1999), it is of great importance to have a detailed analysis of the process. Such knowledge decreases the semantic gap between the high level G/Q/M goals and the questions which need to be asked in building the "G/Q/M tree" (which is a directed acyclic graph). Process improvement may be done to control the negative aspects of process diversity. In this case, our classification framework can help in G/Q/M analysis by relating a G/Q/M goal to an improvement action.

A process is executed within a process environment. The environment should provide mechanisms to address the issues related to process diversity. Our classification can provide additional insight into such mechanisms.

In the following, some of the actions are outlined which can be taken to counteract the negative impacts of each type of process diversity.

6.5.1 Latitudinal Process Diversity

Whenever partners interact, it is important to build a good *co-operative atmosphere* where partners can work together as a "virtual team", joined by common goals (Wells and Harrison, 2001). Participating teams should spend time at the beginning of a project to understand each other's processes, vocabulary, tools and perspective. For example, one development team could participate in seminars or workshops sponsored by the other team. This will also help to break down barriers stemming from cultural differences, creating a better working environment.

Interface issues should be identified and resolved at the beginning of a project rather than reactively as problems arise. Anderson et al. (2001) have proposed a similar idea for handling usability aspects at the start of a project. These issues include not only the functional interfaces required by the software system but also the in-process interfaces between project partners (e.g. for exchanging documents). Shared commitments to using open standards like HTML and XML for data exchange can help both cross-cultural and cross-platform document exchanges.

It is very important that partners be flexible about the detailed definition of processes and should try to tolerate minor differences. For example, one partner's processes cannot usually be imposed on another partner without creating problems. One way to achieve the necessary flexibility could be a *periodic review process for processes*, which should include a method for reaching consensus. This should be compounded by a constant high level of communication, as recommended by Herbsleb and Grinter (1999).

6.5.2 Longitudinal Process Diversity

When developing software, one should be aware that in the future, it might be reused in different projects and contexts. In order to minimise problems stemming from longitudinal process diversity the following points should be kept in mind:

Scalability: this mostly affects the software design process. Aim for a good object-oriented structure, avoiding global functions.

Portability: use of ISO and other standards and minimise the use of non-portable soft- and hardware functions.

Generality: use as few prior assumptions on input and output data as possible, and use abstract concepts during design (e.g. when programming a People Tracker, find a layer of abstraction for tracking methods—the system might be used to track animals in the future).

Interoperability: use open standards (e.g. XML) for data exchange. This reduces platform dependencies for data, documents and configuration information.

Maintainability: probably the most important point, and connected to several of the above. The following measures are recommended to attain a high level of maintainability:

- carefully choose, document and follow a suitable software process to achieve a high *software process consistency over time.*
- create all software artefacts (e.g. programmer's and user manuals) and keep them synchronised with the implementation.
- do refactoring (Demeyer et al., 2000) "on the fly". This can involve incremental reverse engineering using design patterns (Fowler et al., 1999).
- use metrics periodically to monitor design quality.

In order to avoid problems arising from longitudinal process diversity it is particularly important allocate enough resources for software quality and related tasks.

6.6 Conclusions

In a piece of software the size and age of the Reading People Tracker process diversity can easily occur. This is especially so in academic software projects where software processes are usually not well defined and therefore vary with the different people and different research projects a piece of software might be developed and used in. Process diversity thus covers a wide field, occurring in many situations and forms.

A way has been presented to classify process diversity into *longitudinal* and *latitudinal* categories. By studying the case of the Reading People Tracker project which exhibits both types of process diversity, it has been shown how the effects on the quality of products and processes differ with the type of process diversity encountered.

The effects of *longitudinal process diversity* are most relevant where software components or products are reused within significantly different processes. This implies that software engineers should try to anticipate the *possibility* of component reuse within different processes, even when the detailed *form* of reuse cannot be predicted.

The effects of *latitudinal process diversity* are likely to be felt most strongly where "virtual teams" are created for a specific project, especially if the participants are drawn from contrasting traditions of software development and other processes.

It is hoped that understanding the differences between these kinds of process diversity will help software engineers and project managers to assess their implications for process and product quality, and their impact on risk management in projects. Furthermore, relating process diversity to broader concepts of software evolution provides a linkage between fine-grained process improvements and larger-scale, organisational models of the co-evolution of software and business processes.

Chapter 7

Validation of the New People Tracker

> In this chapter the Reading People Tracker is validated using test data from a surveillance camera in an underground station. It is demonstrated that robust realtime tracking of people can be achieved with the new tracking system using standard PC hardware.

After demonstrating the functionality of the Reading People Tracker in Section 4.4 it will now be validated to determine whether it is suitable for robust long-term tracking of people in an underground station.

7.1 Experimental Setup

7.1.1 Test Data

The video sequence used for these experiments was collected at the YZER underground station in Brussels. The feed from the analogue video camera was recorded on analogue `s-vhs` video tape and subsequentially digitised at the original PAL resolution of 768 × 576 pixels. Digitisation was done at a frame rate of 5 frames per second (fps). The 1350 frames of the sequence, ranging from numbered frames 280 to 1629, are therefore equivalent to 4:30 minutes of video. The digitised images were compressed using JPEG image compression at a compression ratio of approximately 13 : 1 (file size 34 KB). Camera 02, which was used in these experiments, is a monochromatic (greyscale) camera at a low height from the ground viewing platform 2. There is a time overlay in the top left corner of the video image. Figure 7.1 shows an image from the video sequence.

The camera was calibrated using 3D point measurements from images with a person holding a measuring rod. From this a 3×4 projection matrix P was determined

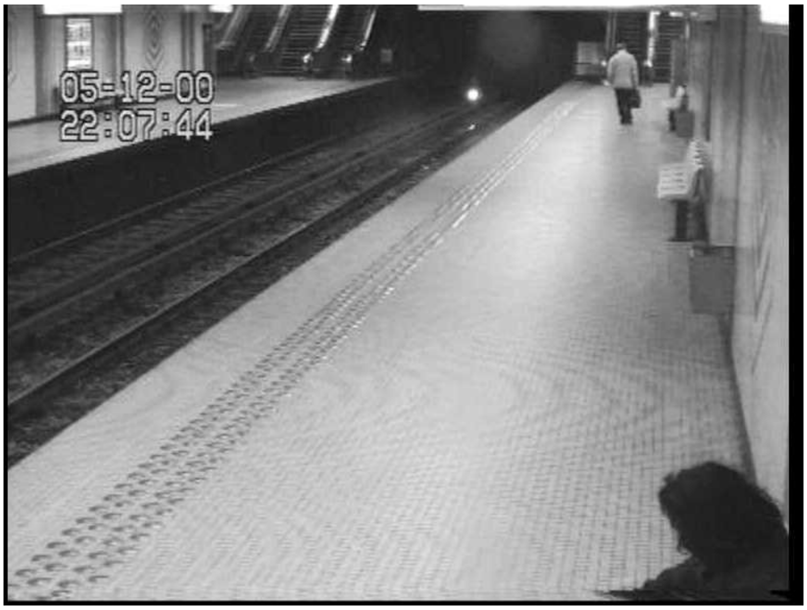


Figure 7.1: View from the YZER station surveillance camera 02

which translates homogeneous world coordinates to homogeneous image coordinates. Thus for a point (x, y, z) in world coordinates with a corresponding image point (u, v) the equation

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{P} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{7.1}$$

holds up to scale. The general accuracy of the calibration data is fair while naturally large inaccuracies occur in areas near the scene's vanishing point in the image.

7.1.2 Configuration of the People Tracker

The Reading People Tracker was run with all four modules enabled: Motion Detector, Region Tracker, Head Detector and Active Shape Tracker. After adjusting the configuration parameters to the sequence the tracker was run in batch mode (no user input, no screen output). Video images with overlayed results, an XML file with tracking output and additional text output were written to hard disk.

In the given video sequence, the image quality is not very good and in some areas of the image the contrast of people against the background is very low. Therefore the Motion Image was processed before extracting moving regions from it. The filter that was used is a simple dilation filter (Kasturi and Trivedi, 1990) which fills holes in the motion image which are up to 1 pixel in size. Thus blobs which are very close together are connected and extracted in one moving area.

The model depth (PCA dimension) used for the Active Shape Tracker is 10, catmeaning that only the first 10 PCA modes for the 64-dimensional shape model are varied to fit the contour shape to measurements in the image[1].

7.2 Experimental Results

Figures 7.2 and 7.3 show key frames from three important passages of the sequence. These key tracking events will be briefly explained here. Section 7.3 below will summarise the results.

In the video images shown in the Figures 7.2 and 7.3 tracking results have been visualised by drawing the bounding boxes of every region tracked by the Region Tracker and by drawing the contour of the person tracked by the Active Shape Tracker. When objects are drawn in the same colour, their identification record is the same. The output from the Head Detector is used to initialise shapes in the Active Shape Tracker, but the detected heads are not drawn here.

Frames shown in Figure 7.2, page 108:

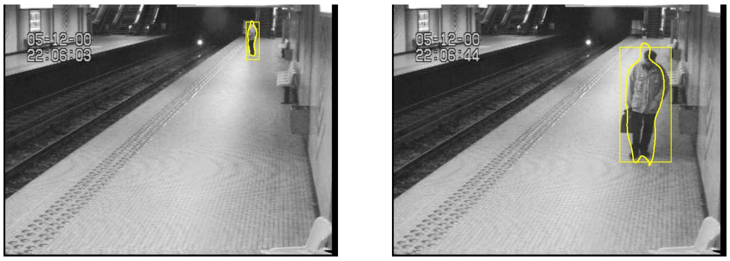
Frame 283 One person, “A”, is standing at the far end of the platform. Although still very small in the image, person A, marked in yellow, is picked up by both Region Tracker and Active Shape Tracker. The Active Shape is initialised from the Region Tracker and Head Detection output. Thus the two tracks get assigned the same identity by the two trackers, but they are tracked separately.

Frame 490 Person A has come closer to the camera. It can now be seen how the Active Shape Tracker approximates the person’s outline in the image. While the person’s general appearance is modelled well, the model does not explain the bag carried by the person and the fact that he is leaning slightly to the right.

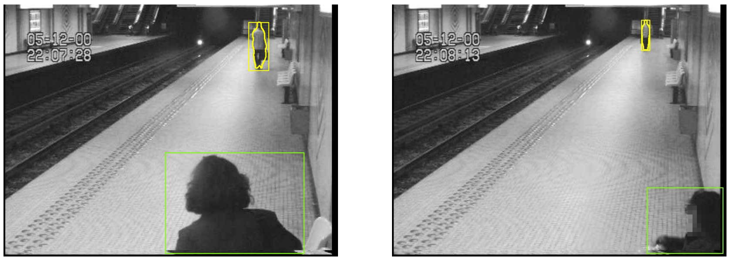
Frame 706 Person A has walked back to the far end of the platform. Meanwhile, a second person, “B”, has entered the scene from the front, turning around to sit down on a bench close to the camera. Only the upper part of his body can be seen, and the Active Shape Tracker cannot initialise a shape. However, the Region Tracker keeps track of the person (marked by a green bounding box).

---

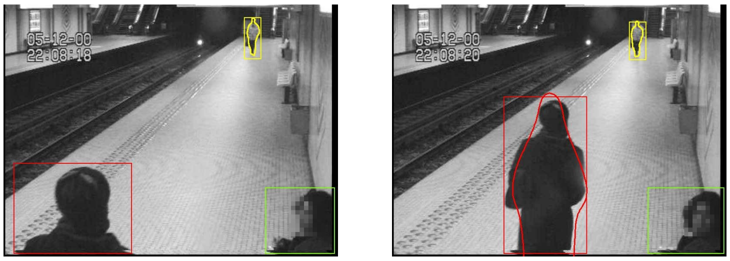
[1] The results do not depend much on the model depth; using a maximum model depth of 5 yields very similar results.


05-12-00
22:06:03
05-12-00
22:06:44

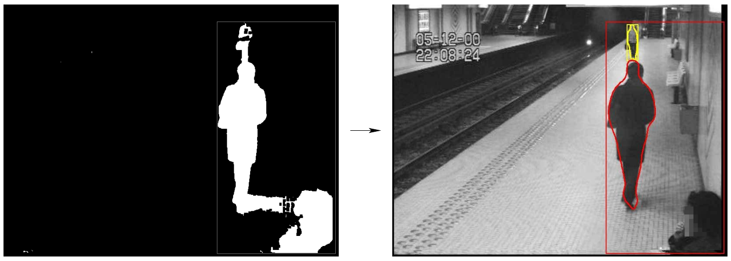
(a) Frame 283 (b) Frame 490


05-12-00
22:07:28
05-12-00
22:08:13

(c) Frame 706 (d) Frame 933


05-12-00
22:08:18
05-12-00
22:08:20

(e) Frame 960 (f) Frame 970


05-12-00
22:08:24

(g) Frame 986, Motion Image (h) Frame 986

Figure 7.2: Frames 283 through 986 (time: 0:00 to 2:20.6 minutes)

Frame 933 At this point, person A has moved so far away from the camera that the motion image does not show him clearly. Consequently the Region Tracker does not have a measurement for this frame. However, the Active Shape Tracker keeps tracking the person, and the Region Tracker uses part of the output from the Active Shape Tracker (the bounding box of the contour) to adjust and keep the track.

Frame 960 2:15 minutes into the sequence a woman, person “C”, enters the scene from the front. Her head is tracked by the Region Tracker, drawn in red. The Active Shape Tracker fails to initialise the track because only a small part of her outline is visible at this stage.

Frame 970 A few frames later, the Active Shape Tracker has picked up the track of person C, assigning to it the identity and tracking history of the corresponding region which has been tracked by the Region Tracker for a while.

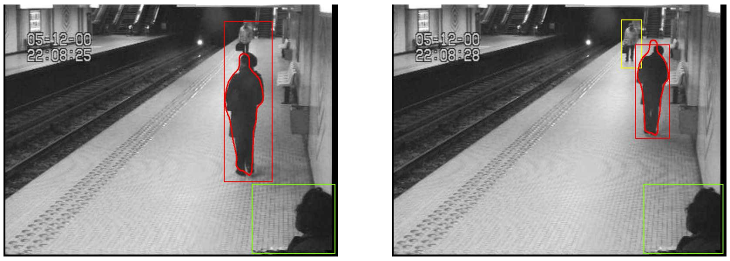
Frame 986 At this point in time, the three people come so close in the image that they are extracted as one large region from the Motion Image. As a result, the track of the sitting person, B, is not detectable for a short period of time. Persons A and C are still tracked. The reason why only two of the tracks are recovered is that the Region Splitting feature in the Region Tracker currently only handles the case of splitting a region into two, not more.

Frames shown in Figure 7.3, page 110:

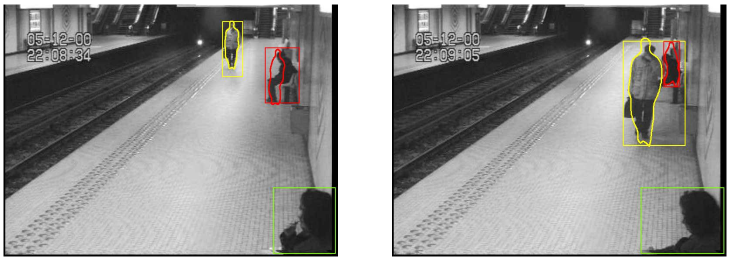
Frame 994 After a few frames, person B is detected by the Motion Detector again, and the Region Tracker takes over the track. It realises that this is the same person seen previously and consequently assigns the same identity and tracking history. This is possible because person B’s track was only lost for a few frames, and the person has not moved much in the meantime. Person C is still tracked well while person A is too much occluded to be detected at this time.

Frame 1010 The Region Tracker has again taken up track of person A. Thereby all people are tracked again with their original identities.

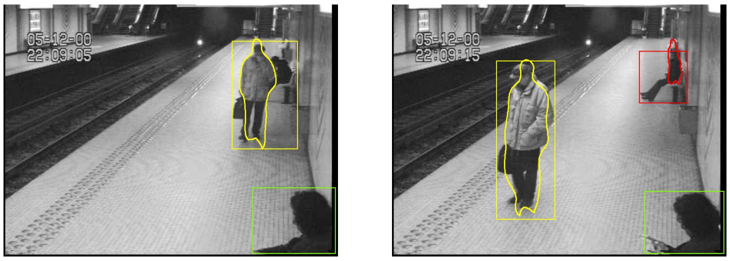
Frame 1039 Person C sits down on a bench. The Active Shape Tracker is not trained to recognise this deformation of a person’s outline and thus has difficulties to detect the correct outline. The current shape estimate, although not backed up by many image measurements, is drawn here to illustrate this. The Region Tracker keeps tracking person C even when seated since it is not specialised to track objects with particular appearances only.


05-12-00
22:08:25
05-12-00
22:08:28

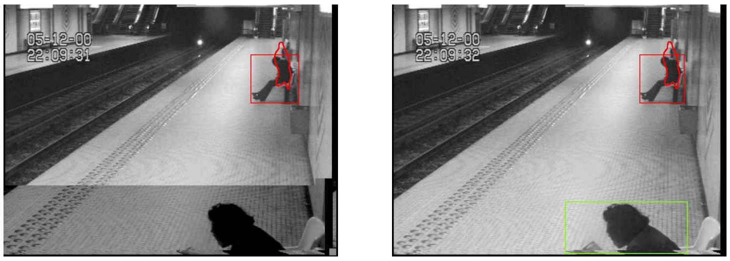
(a) Frame 994 (b) Frame 1010


05-12-00
22:08:34
05-12-00
22:09:05

(c) Frame 1039 (d) Frame 1191


05-12-00
22:09:05
05-12-00
22:09:15

(e) Frame 1193 (f) Frame 1243


05-12-00
22:09:31
05-12-00
22:09:32

(g) Frame 1325 (h) Frame 1326

Figure 7.3: Frames 994 through 1326 (time: 3:18.8 to 4:25.2 minutes)

Frame 1191 Person A has come even closer to person C in the image and partly occludes C. The Active Shape Tracker's estimate of C's contour therefore degrades further. The upper part of C's body is now estimated to be a small person standing on the bench.

Frame 1193 When the occlusion of person C is maximal she is not detected and the track lost for a short time. The other people are tracked fine.

Frame 1243 As all people are again separate in the image the tracks are resumed with correct identities. Person A is walking toward the camera and as he gets closer some inaccuracies of the model (e.g. the head) again become apparent. However, the track is closely maintained by the Active Shape Tracker. Person C is still seated. The Region Tracker shows a correct track of C while the Active Shape Tracker again exhibits the inadequacy of its shape model for sitting people.

Frame 1325 Person A has left the scene. At this point in time the video signal is affected by some noise which causes the lower image to be distorted. Consequentially person B is not detected for one frame.

Frame 1326 The track of person B is again taken up with the correct identity.

7.3 Summary and Conclusions

7.3.1 Tracking Performance

The results show a good overall performance of the Reading People Tracker. Over 4:30 minutes all people have been tracked with correct identities, resulting in long-time tracking output which can be used for behaviour analysis.

Figure 7.4 shows detailed statistics on the performance of the Region Tracker (RT) and the Active Shape Tracker (AST). Separate graphs for each person show when the person was tracked by each of the trackers. The axes are scaled to show the time when the person in question is visible in the image. This also means that the frame numbers are not aligned between the three graphs. Short annotations in the graphs point out the reasons why tracking fails in a few situations; these situations were described in more detail above.

Several times during the sequence, the Region Tracker kept tracking a person when the Active Shape Tracker failed. This happened in circumstances where the active shape model had to cope with people's shapes it had not been trained to recognise (e.g. people sitting down) or when insufficient part of a person's outline was visible. The Active Shape Tracker itself aided the Region Tracker in situations where the Motion Detector could not extract clear moving images of people. In these

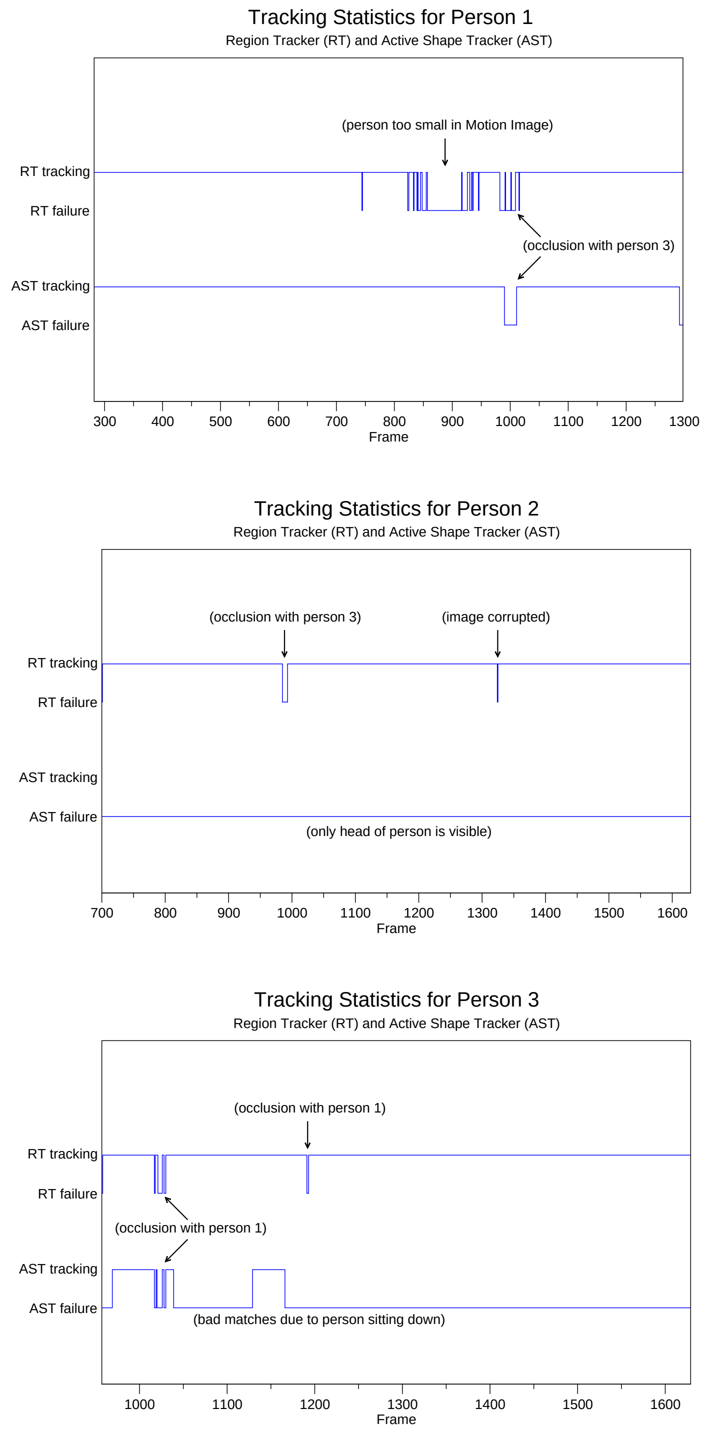


Figure 7.4: Performance of Region Tracker and Active Shape Tracker

situations the Region Tracker has used the output from the Active Shape Tracker (bounding box of current shape estimate) for tracking, if available. In Figure 7.4 this is still marked as “RT failure”.

The combined tracker has tracked all people reliably. In some situations tracks were lost for a few frames by one or even both of the individual trackers. However, all these tracks were re-established at a later stage by at least one of the trackers.

It can also be observed that it clearly helps to keep predicting a track for a few frames when it is not detectable. Once the track is re-gained the identity and tracking history can thus be established. This was helpful in a few cases especially when there is occlusion or a corrupted image.

7.3.2 Processing Speed

In the experiment described above the processing speed was 4.34 frames per second (fps). The processing includes reading video images in JPEG format from hard disk and uncompressing them using a standard system library. Tracking results were written out in XML format, but no images were displayed to screen. The image dimensions were the standard PAL format (768 × 576 pixels). The tracker was run on a PC equipped with two CPUs, each one a 1 GHz `Pentium III`, running `GNU/Linux` version 2.4.19. Although two CPUs are installed in the system, essentially only one of them was used, and the other one was 99 % idle. While the Reading People Tracker is multi-threaded it uses one processing thread per camera. In the experiment only one camera was used and therefore in this situation multi-threading did not have a substantial effect.

One of the reasons why the target frame rate of 5 fps was not achieved is that the Motion Image was post-processed with a dilation filter. This was necessary because of low contrast of the people to the background in some areas of the image. When disabling motion image dilation the frame rate increases to 5.49 fps which is more than sufficient.

In the `ADVISOR` system, the image dimensions are reduced to a quarter of the above size in pixels—that is, 384 × 288 pixels. Most of the CPU time is used for image processing operations which are done for every pixel. Using these reduced image sizes therefore significantly increases processing speed. This has been confirmed in separate experiments.

7.3.3 Conclusions

It has been shown that the Reading People Tracker can successfully track people in camera images over an extended period of time. The combination of a Region Tracker with an Active Shape Tracker introduces a redundancy which more than once prevented track loss in difficult situations, e.g. occlusion.

The experiment has also shown that processing speed depends on image quality. Low image quality and low contrast necessitate post-processing of images with image filters. These image filters increase tracking robustness but also slow down processing. This is particularly significant when images with large dimensions are used because these operations are done for every pixel in the image.

Chapter 8

Conclusions

8.1 Summary of Work

The work presented in this thesis was concerned with building a people tracking application for automated visual surveillance of an underground station. The Reading People Tracker which has been designed and implemented for this purpose solves the task of tracking people reliably and in realtime. The new people tracker has been integrated into the automated visual surveillance system `ADVISOR` for which it has been adapted and extended from an existing tracker, the *Leeds People Tracker* by Adam Baumberg.

In **Chapter 3** a method is introduced to filter video images in a way which minimises the influence of image noise and bad image quality on people tracking algorithms. Tests on images with typical image noise from an underground surveillance system show how edge contrast has been increased by the proposed method.

Chapter 4 describes the tracking algorithms used in the Reading People Tracker. The task of people tracking is carried out by four detection and tracking modules: a Motion Detector, a Region Tracker, a Head Detector and an Active Shape Tracker. Their combination and co-operation introduces redundancy and thereby a greater robustness and generality of the tracker. The new algorithms have been shown to aid tracking in a demonstration of the new features and co-operation techniques.

An important aspect of the Reading People Tracker is the maintainability of its software. Using software re-engineering techniques the tracker was made highly maintainable, making it possible to adapt, extend and improve the software with ease. In **Chapter 5** this re-engineering process and its influence on the maintainability of the people tracker was studied from a software engineering perspective.

Chapter 6 investigates the software engineering aspect of the Reading People Tracker in more detail. It focuses on the software processes which have been used to maintain the software over its lifetime of 10 years. It was shown that not only the quality of a software process but also the diversity of software processes has a significant impact on software. A new classification scheme for process diversity into two broad types, *latitudinal* and *longitudinal process diversity* was presented. The influence of both types on the quality of software was examined.

In **Chapter 7** the Reading People Tracker was validated against test data collected from an underground station in Brussels. The experimental results shows that the new Reading People Tracker is suitable for tracking people over a long period of time in a robust and reliable manner.

8.2 Discussion

The Reading People Tracker which has been developed in this thesis uses a combination of four co-operating detection and tracking modules to track people in camera images. Each of the modules is of medium to low complexity. The approach of combining multiple algorithms and letting them exchange their results succeeds in overcoming some limitations which individual modules have. At the same time, the negative aspects of more complex algorithms like full 3D models of human beings are avoided. The overall people tracker has been shown to track individuals robustly in the presence of occlusion and low image quality.

Existing problems for the tracker are situations in which the background model fails, and strong occlusion. The first problem occurs when there are fast lighting changes, e.g. when there are clouds in the sky in outdoor surveillance scenes. Indoors this can happen when a train arrives and the camera's automatic gain control (AGC) adjusts to the new overall brightness of the image. The second problem, occlusion, becomes apparent when groups of people are tracked. The strategy of the tracker is to separate individuals within a group using Head Detection and the Active Shape Model, and subsequentially Region Splitting. When these measures fail, the group is only tracked as a single moving object by the Region Tracker.

8.3 Future Work

By carefully designing the structure and implementation of the Reading People Tracker the software has been made highly maintainable. It can easily be scaled, extended and otherwise adapted for different tracking tasks in future projects. One example would be the addition or replacement of a detection or tracking module. Documentation is available on how to understand the design of the people tracker and then carry out these tasks (Bouffant et al., 2002). There follows are a few suggestions for future changes and additions.

Future applications of the Reading People Tracker may require a different Motion

Detector. This is particularly important if the tracker is to be used in an outdoor surveillance situation. In Section 2.1.2, motion detection techniques with more sophisticated background models have been introduced. A trade-off has to be made between performance and processing speed since these complex motion detectors are very CPU intensive.

This thesis has not tried to solve the problem of tracking dense groups of people—that is, groups where there is heavy occlusion between people. For the `ADVISOR` system this problem is not so important as most events that are to be detected (e.g. vandalism) involve only a few people and do not usually take place when many of the people are around.

The design of the Reading People Tracker allows many detection and parameters to be changed and "tuned" to a particular situation or scene. The total number of configuration options is 115, although about a third of these options are file names and other options which are not concerned with tracking. The number of tracking parameters which have been modified for the test runs presented in this thesis does not exceed 10. It would be an interesting project to find out which parameters can be left constant and which can be derived from other parameters. This would have to be done with a special statistical tool which runs the tracker automatically with varying parameters. It also necessitates the availability and use of "ground truth"—that is, known tracking results—for a large set of video sequences.

List of Figures

1.1 The task of detecting and tracking people 2
1.2 People Tracking as one of six subsystem of `ADVISOR` 2
1.3 Structure of the *Reading People Tracker* 3

2.1 Example image with three blobs and a region 8
2.2 Example Motion Image 9
2.3 People tracked by the *Leeds People Tracker* 12
2.4 Detection hierarchy within the *Leeds People Tracker* 13
2.5 Edge search for shape fitting 14
2.6 The W^4 system: Detected body parts 17
2.7 Gavrila's 3D tracker: Views of recovered torso and head 20
2.8 Sidenbladh's 3D people tracker: Superimposed model on tracked person 21
2.9 Overview of the six subsystem of the `ADVISOR` system 25
2.10 Incremental stages and requirements within the `ADVISOR` project . . . 26

3.1 The path of an image from the video camera to the tracker 34
3.2 The way a video image is used by the People Tracker 36
3.3 Local search for edges during shape fitting 37
3.4 Alternative methods to produce the Difference Image 38
3.5 Video image used for colour filtering experiments 39
3.6 Chrominance channels C_B and C_R of the original Video Image 40
3.7 Video Image and Colour Difference Image used for Experiments . . . 42
3.8 Filtering the Colour Difference Image 43
3.9 Differencing filtered Video and Background Images 45

4.1 People Tracking as one of six subsystem of `ADVISOR` 48
4.2 Overview of the four modules of the *Reading People Tracker* 49
4.3 Problem during motion detection, solved by Region Splitting 51
4.4 The Algorithm of the *Reading People Tracker* 53
4.5 Head detected by the Head Detector 56
4.6 Interaction between tracking modules 59
4.7 Frame 2 (Motion and Video Images); Frames 3 and 4 (Video Images). 63

4.8 Frames 39 and 55 (Video Images). 64
4.9 Initialisation and refined tracks, frame 6 65

5.1 Data flow within the `ADVISOR` integrated system 73
5.2 Sequence of work carried out during re-design 75
5.3 *Reading People Tracker*: Size in Lines of Code over Time 78
5.4 Use Case for the People Tracker (in Standalone/Development Mode) 79
5.5 Use Case for the People Tracker (as a Subsystem of `ADVISOR`) 80
5.6 Software Packages of the People Tracker 81
5.7 High level Sequence Diagram of the People Tracker within `ADVISOR` . 82

6.1 Software Process Model, Stereotype 1—Low Propensity for Evolution 91
6.2 Software Process Model, Stereotype 2—High Propensity for Evolution 92
6.3 People Tracking as one of six subsystems of `ADVISOR` 93
6.4 Main development and maintenance phases of the Reading People Tracker 94
6.5 Latitudinal and Longitudinal Process Diversity 96

7.1 View from the YZER station surveillance camera 02 106
7.2 Frames 283 through 986 (time: 0:00 to 2:20.6 minutes) 108
7.3 Frames 994 through 1326 (time: 3:18.8 to 4:25.2 minutes) 110
7.4 Performance of Region Tracker and Active Shape Tracker 112

List of Tables

5.1 Measurements for different versions of the *Reading People Tracker* . . 74
5.2 Distribution of the Maintenance Effort 77
5.3 Maintenance effort by category, compared to the industry average . . 77

6.1 Measurements of the People Tracker over Time 99

Bibliography

Jean Anderson, Francie Fleek, Kathi Garrity and Fred Drake. Integrating usability techniques into software development. *IEEE Software*, 18(1):46–53, January/February 2001.

Algirdas Avizienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, December 1985.

Harry G Barrow, Jay M Tenenbaum, Robert C Bolles and Helen C Wolf. Parametric correspondence and Chamfer matching: Two new techniques for image matching. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 659–663, 1977.

Adam M Baumberg. *Learning Deformable Models for Tracking Human Motion.* PhD thesis, School of Computer Studies, University of Leeds, Leeds, UK, October 1995. `ftp://ftp.comp.leeds.ac.uk/comp/doc/theses/baumberg.ps.gz`.

Adam M Baumberg. Hierarchical shape fitting using an iterated linear filter. In *Proceedings of the Seventh British Machine Vision Conference (BMVC96)*, pages 313–322. BMVA Press, 1996. `http://www.bmva.ac.uk/bmvc/1996/baumberg_1.ps.gz`.

Adam M Baumberg. Personal communication, November 2002.

Adam M Baumberg and David Hogg. An adaptive eigenshape model. In *Proceedings of the Sixth British Machine Vision Conference (BMVC95)*, volume 1, pages 87–96, 1995. `http://www.scs.leeds.ac.uk/vision/proj/amb/Postscript/bmvc2.ps.Z`.

Kent Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, Reading, USA, 1999.

Alessandro Bianchi, Danilo Caivano, Filippo Lanubile, Francesco Rago and Giuseppe Visaggio. Distributed and colocated projects: A comparison. In *Proocedings of the 7th Workshop on Empirical Studies of Software Maintenance (WESS01)*, pages 65–69, 2001.

Tugdual Le Bouffant, Nils T Siebel, Stephen Cook and Steve Maybank. The Reading People Tracker version 1.12: Reference manual. Technical Report RUCS/2002/TR/11/001/A, Computational Vision Group, Department of Computer Science, The University of Reading, Reading, UK, November 2002.

François Brémond and Monique Thonnat. Tracking multiple non-rigid objects in a cluttered scene. In *Proceedings of the 10th Scandinavian Conference on Image Analysis (SCIA '97), Lappeenranta, Finland*, volume 2, pages 643–650, June 1997. `http://www-sop.inria.fr/orion/Publications/Articles/PostScript/SCIA97.ps`.

Qin Cai, Amar Mitiche and J K Aggarwal. Tracking human motion in an indoor environment. In *Proceedings of the 2nd International Conference on Image Processing (ICIP'95)*, pages 215–218, 1995.

Erran Carmel. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, New Jersey, USA, 1st edition, 1999.

Fabiano Cattaneo, Alfonso Fuggetta and Luigi Lavazza. An experience in process assessment. In *Proceedings of the 17th International Conference on Software Engineering (ICSE-17), Seattle, USA*, pages 115–121, April 1995.

Brian W Chatters, Meir M Lehman, Juan F Ramil and Paul Wernick. Modelling a software evolution process: A long-term case study. *Software Process Improvement and Practice*, 5(2–3):91–102, June–September 2000.

Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

Nicolas Chleq, François Brémond and Monique Thonnat. Image understanding for prevention of vandalism in metro stations. In Carlo S. Regazzoni, Gianni Fabri and Gianni Vernazza, editors, *Advanced Video-based Surveillance Systems*, volume 488 of *The Kluwer International Series in Engineering and Computer Science*, chapter 3.2, pages 106–116. Kluwer Academic Publishers, Boston, USA, November 1998.

James L Crowley and François Bérard. Multi-modal tracking of faces for video communications. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 640–645, 1997. `http://iihm.imag.fr/publs/1997/CVPR97_MMTracking.ps.gz`.

Serge Demeyer, Stéphane Ducasse and Oscar Nierstrasz. Finding refactorings via change metrics. In *Proceedings of the 2000 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2000)*, pages 166–177, October 2000.

Raymond Dion. Process improvement and the corporate balance sheet. *IEEE Software*, 10(4):28–35, July/August 1993.

Matthew Lee Domsch and Stephen R Schach. A case study in object-oriented maintenance. In *Proceedings of the 1999 International Conference of Software Maintenance (ICSM '99), Oxford, UK*, pages 346–352, August 1999.

John C Doppke, Dennis Heimbigner and Alexander L Wolf. Software process modeling and execution within virtual environments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(1):1–40, January 1998.

Ahmed Elgammal, David Harwood and Larry Davis. Non-parametric model for background subtraction. In David Vernon, editor, *6th European Conference on Computer Vision (ECCV 2000), Dublin, Ireland*, pages 751–767. Springer Verlag, 2000.

Norman E Fenton and Shari Lawrence Pfleeger. *Software Metrics.* PWS Publishing Company, Boston, USA, 2nd edition, 1996.

Adrian Ford and Alan Roberts. Colour space conversions. Available at `http://www.inforamp.net/~poynton/PDFs/coloureq.pdf`, August 1998.

David A Forsyth and Jean Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, New Jersey, USA, 2003.

Martin Fowler, Kent Beck, John Brant, William Opdyke and Don Roberts. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, Reading, USA, 1999.

Dariu M Gavrila. *Vision-based 3D Tracking of Humans in Action.* PhD thesis, Department of Computer Science, University of Maryland, College Park, USA, 1996. `http://www.gavrila.net/Publications/thesis_ps.zip`.

Dariu M Gavrila and Larry S Davis. Tracking of humans in action: A 3-D model-based approach. In *ARPA Image Understanding Workshop, Palm Springs, USA*, pages 737–746, February 1996. `http://www.gavrila.net/iuw.ps.Z`.

Neil A Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press, Cambridge, UK, 1999.

Ronald L Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, June 1972.

David Gries. *The Science of Programming.* Springer-Verlag, New York, USA, 1981.

Ismail Haritaoglu, David Harwood and Larry S Davis. W^4: Real-time surveillance of people and their actions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):809–830, August 2000.

James Herbsleb, Anita Carleton, James Rozum, Jane Siegel and David Zubro. Benefits of CMM-based software process improvement: Initial results. Technical Report CMU/SEI-94-TR-013, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, August 1994.

James D Herbsleb and Rebecca E Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999), Los Angeles, USA*, pages 85–95, May 1999.

Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

David Hogg. Model-based vision: A program to see a walking person. *Image and Vision Computing*, 1(1):5–20, February 1983.

Watts Humphrey. Introduction to software process improvement. Technical Report CMU/SEI-94-TR-007, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, August 1992. Revised version, August 1993.

IEEE Std-1471-2000(2000). *IEEE Std-1471-2000: Recommended Practice for Architectural Description of Software Intensive Systems.* IEEE Computer Society, Piscataway, USA, 2000.

POSIX 1003.1c-1995(1995). *IEEE POSIX 1003.1c-1995.* IEEE Standards Association, 1995.

ISO/IEC 9126(1991). *ISO/IEC 9126.* International Organization for Standardization, Geneva, Switzerland, 1991.

ISO/IEC 12207(1995). *ISO: Information Technology—Software Life Cycle Processes. ISO/IEC 12207:1995.* International Organization for Standardization, Geneva, Switzerland, 1995.

ISO/IEC 9899(1999). *ISO: Programming languages—C. ISO/IEC 9899:1999(E).* International Organization for Standardization, Geneva, Switzerland, 1999. `http://www.dkuug.dk/JTC1/SC22/WG14`.

Neil Johnson. *Learning Object Behaviour Models.* PhD thesis, School of Computer Studies, University of Leeds, Leeds, UK, September 1998. `http://www.scs.leeds.ac.uk/neilj/ps/thesis.ps.gz`.

Rangachar Kasturi and Mohan M Trivedi, editors. *Image Analysis Applications*. Number 24 in Optical Engineering. Marcel Dekker, New York, USA, 1990.

Sohaib Khan, Omar Javed, Zeeshan Rasheed and Mubarak Shah. Human tracking in multiple cameras. In *Proceedings of the 8th IEEE International Conference on Computer Vision (ICCV 2001), Vancouver, Canada, July 9–12, 2001*, pages 331–336, July 2001. `http://www.cs.ucf.edu/~vision/ln2Vis5/handoff-iccv.pdf`.

Thomas G Lane. *IJG JPEG Library: System Architecture*. Independent JPEG Group, 1991–1995. Part of the Independent JPEG Group's JPEG software documentation, see `http://www.ijg.org/`.

Meir M Lehman. The programming process. IBM Research Report RC 2722, IBM Research Center, Yorktown Heights, USA, 1969.

Meir M Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, September 1980.

Meir M Lehman, Dewayne E Perry, Juan F Ramil, Władysław M Turski and Paul Wernick. Metrics and laws of software evolution—the nineties view. In *Proceedings of the Fourth International Conference on Software Metrics (Metrics 97)*, pages 20–32, November 1997.

Bennet P Lientz, E Burton Swanson and G E Tompkins. Characteristics of application software. *Communications of the ACM*, 21(6):466–471, June 1978.

Chris Lilley, Fenqiang Lin, W T Hewitt and T L J Howard. ITTI computer graphics and visualization—Colour in computer graphics. Student notes, Computer Graphics Unit, Manchester Computing Centre, University of Manchester, Manchester, UK, December 1993.

Alan J Lipton, Hironobu Fujiyoshi and Raju S Patil. Moving target classification and tracking from real-time video. In *Proceedings of the DARPA Image Understanding Workshop (IUW'98), Monterey, USA*, pages 129–136, November 1998. `http://www.cs.cmu.edu/~vsam/Papers/wacv98_tracking.ps.gz`.

Tim Littlefair. *An Investigation into the Use of Software Code Metrics in the Industrial Software Development Environment*. PhD thesis, Faculty of Communications, Health and Science, Edith Cowan University, Perth, Australia, 2001.

Radu Marinescu. Detecting design flaws via metrics in object-oriented systems. In *Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Language and System (TOOLS USA 2001)*, pages 173–182, July/August 2001.

EventHelix.com Inc. Issues in realtime system design. Article in an online collection, 2000–2002. `http://www.eventhelix.com/RealtimeMantra/IssuesInRealtimeSystemDesign.htm`. Visited on Sat Nov 9 19:37:58 GMT 2002.

A J Mitchell-Jones. Personal communication, February 2002.

Alan Moore and Tony Backwith. Development tools for real-time systems. *Real-Time Magazine*, 7(1):33–37, Quarter 1 1999. `http://www.realtime-info.com/magazine/99q1/1999q1_p033.pdf`.

Nuria Oliver, Alex P Pentland and François Bérard. Lafter: Lips and face real time tracker. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 123–129, 1997. `http://iihm.imag.fr/publs/1997/CVPR97_LAFTER.ps.gz`.

Joseph O'Rourke and Norman Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(6):522–536, November 1980.

David Lorge Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering (ICSE-16), Sorrento, Italy*, pages 279–287, May 1994. Invited Plenary Talk.

Mark C Paulk. Models and standards for software process assessment and improvement. In Robin B Hunter and Richard H Thayer, editors, *Software Process Improvement*, chapter 1, pages 1–36. IEEE Press, Piscataway, USA, November 2001.

Mark C Paulk, Bill Curtis, Mary Beth Chrissis and Charles V Weber. Capability maturity model, version 1.1. *IEEE Software*, 10(4):18–27, July 1993.

Rational Rose 2000e(2000). *Rational Rose 2000e.* Rational Software Corporation, Cupertino, USA, 2000.

Paolo Remagnino, Adam M Baumberg, Tom Grove, Tieniu Tan, David Hogg, Keith Baker and Anthony Worrall. An integrated traffic and pedestrian model-based vision system. In Adrian Clark, editor, *Proceedings of the Eighth British Machine Vision Conference (BMVC97)*, pages 380–389. BMVA Press, 1997.

Tamar Richner and Stéphane Ducasse. Recovering high-level views of object-oriented applications from static and dynamic information. In *Proceedings of the 1999 International Conference of Software Maintenance (ICSM '99)*, pages 13–22, August 1999.

Simon Rowe and Andrew Blake. Statistical mosaics for tracking. *Image and Vision Computing*, 14(8):549–564, August 1996.

James Rumbaugh, Ivar Jacobson and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, USA, 1999.

Manoranjan Satpathy, Nils T Siebel and Daniel Rodríguez. Maintenance of object oriented systems through re-engineering: A case study. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002), Montréal, Canada*, pages 540–549, October 2002.

Helen Sharp, Mark Woodman, Fiona Hovenden and Hugh Robinson. The role of 'culture' in successful software process improvement. In *Proceedings of the 25th Euromicro Conference (EUROMICRO '99), Milano, Italy*, volume 2, pages 2170–2176, September 1999.

Hedvig Sidenbladh. *Probabilistic Tracking and Reconstruction of 3D Human Motion in Monocular Video Sequences*. PhD thesis, Institutionen för Numerisk analys och datalogi, Kungliga Tekniska Högskolan (KTH), Stockholm, Sweden, November 2001. `http://www.nada.kth.se/~hedvig/publications/thesis.pdf`.

Hedvig Sidenbladh, Michael J Black and David J Fleet. Stochastic tracking of 3D human figures using 2D image motion. In David Vernon, editor, *6th European Conference on Computer Vision (ECCV 2000), Dublin, Ireland*, pages 702–718. Springer Verlag, 2000. `http://www.cs.brown.edu/people/black/Papers/eccv00.ps.gz`.

Nils T Siebel, Steve Cook, Manoranjan Satpathy and Daniel Rodríguez. Latitudinal and longitudinal process diversity. *Journal of Software Maintenance and Evolution*, 15(1):9–25, January–February 2003.

Nils T Siebel and Steve Maybank. The application of colour filtering to real-time person tracking. In *Proceedings of the 2nd European Workshop on Advanced Video-Based Surveillance Systems (AVBS'2001), Kingston upon Thames, UK*, pages 227–234, September 2001a.

Nils T Siebel and Steve Maybank. On the use of colour filtering in an integrated real-time people tracking system. In Paolo Remagnino, Graeme A Jones, Nikos Paragios and Carlo Regazzoni, editors, *Video Based Surveillance Systems: Computer Vision and Distributed Processing*, chapter 14, pages 167–175. Kluwer Academic Publishers, Boston, USA, 2001b.

Nils T Siebel and Steve Maybank. Real-time tracking of pedestrians and vehicles. In *Proceedings of the 2nd IEEE International Workshop on Performance Evaluation* *of Tracking and Surveillance (PETS'2001), Kauai, USA*, December 2001c. CD-ROM proceedings.

Nils T Siebel and Steve Maybank. Fusion of multiple tracking algorithms for robust people tracking. In Anders Heyden, Gunnar Sparr, Mads Nielsen and Peter Johansen, editors, *Proceedings of the 7th European Conference on Computer Vision (ECCV 2002), København, Denmark*, volume IV, pages 373–387, May 2002.

Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99), Fort Collins, USA*, volume 2, pages 246–252, 1999.

Chris Stauffer, W Eric L Grimson, Raquel Romano and Lily Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'98), Santa Barbara, USA*, volume 1, pages 22–29, June 1998.

David Travis. *Effective Color Displays—Theory and Practice*. Academic Press, London, UK, 1991.

Rini van Solingen and Egon Berghout. *The Goal/Question/Metric Method*. McGraw-Hill, London, UK, 1999.

Brian C Warboys, R Mark Greenwood and Peter Kawalek. Modelling the co-evolution of business processes and IT systems. In Peter Henderson, editor, *Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme*, pages 10–23. Springer Verlag, London, UK, 2000.

Richard C Waters and Elliot Chikovsky. Reverse engineering progress along many dimensions. *Communications of the ACM*, 37(5):23–24, May 1994.

Bruce W Weide, Wayne D Heym and Joseph E Hollingsworth. Reverse engineering of legacy code exposed. In *Proceedings of the 17th International Conference on Software Engineering (ICSE-17), Seattle, USA*, pages 327–331, April 1995.

Moira Wells and Rachel Harrison. The liminal moment. Understanding distributed communication and business processes. In *Proceedings of the Conference on Empirical Assessment in Software Engineering (EASE 2001)*, 2001. Pages unnumbered.

Norman Wilde and Ross Huitt. Maintenance support for object oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1038–1044, December 1992.

XML Schema(2001). *XML Schema Part 0: Primer. W3C Recommendation*. World Wide Web Consortium (W3C), 2001. `http://www.w3.org/TR/xmlschema-0/`.

Anthony Worrall. Personal communication, October 2002.

Christopher Wren, Ali Azarbayejani, Trevor Darrell and Alex Pentland. Pfinder: Real-time tracking of the human body. Technical Report 353, MIT Media Laboratory Perceptual Computing Section, 1995. `http://www-white.media.mit.edu/vismod/publications/techdir/TR-353.ps.Z`, also published in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

Jia Hong Yin, Sergio A Velastin and Anthony C Davies. Measurement of crowd density using image processing. In M J J Holt, Colin F N Cowan, Peter M Grant and William A Sandham, editors, *Proceedings of the VII. European Signal Processing Conference (EUSIPCO-94), Edinburgh, UK*, volume III, pages 1397–1400, September 1994.

Index

N-Version Programming, 88

active shape model, 12
ADVISOR, vii, 2–5, 16, 23–26, 28, 29, 33, 47, 48, 62, 67, 69, 72–74, 77, 79, 80, 82, 84, 93, 95–99, 113, 115, 117
agile processes, 90
ARTiSAN Software Tools, 26

B-spline, 13, 15, 56
Background Image, 35
background model, 8
Baumberg, Adam, 12
Black, Michael, 21
blob, *7*, 8
blue, 32
Bull, 24

Capability Maturity Model, 87
CCCC, 73, 99
Chamfer, 20, 21
chrominance, 32, *32*, 40, 46
CIE, *32*, 41
CMM, 87, 95
Colour space
  HSV, *32*
  RGB, *32*
  $\mathrm{YC}_B\mathrm{C}_R$, *32*
colour space, 31
composite, 32
CORBA, 26

data classes, 72
Davis, Larry, 16, 19
DCT, 27
Demonstrator, 27
Difference Image, 36
dynamic analysis, 76
dynamic information, 71, *71*

E-type, 92
Eklundh, Jan-Olof, 21

FEAST, 92

G/Q/M, 87
Gavrila, Dariu, 19
Goal/Question/Metric method, 87
god classes, 72
Graham Scan, 17
green, 32
greyscale, 32

Haritaoglu, Ismail, 16
Harwood, David, 16
Hogg, David, 12
HSI, 32
HSL, 32
hue, 32, *32*, 41, 42, 44

Intel, 19, 62
intensity, 32

JPEG, 27, 32

latitudinal process diversity, 88
Leeds People Tracker, 3, 12, 15, 16, 23, 28, 29, 39, 41, 48, 50, 56, 57, 61, 64, 66, 115
lightness, 32

local image filter, *37*
London Underground Ltd, 5
luminance, 32, *32*, 41

Mahalanobis optimal search direction, 14, 15
maintenance,
    see software maintenance
maintenance effort, 72
mapping function, *38*
median filter, 8
Microsoft, 26
model, 9
    classification of human models, 10–11
    complexity, 9–10
    dimensionality, *9*
    parameterised, 9, *9*
Motion Detection, 7–8
motion detector, 50
Motion Image, 8, 36
motion image, 8
MPEG, 32

P-type, 91
PCA, 12
process, *86*
process diversity, 86
    latitudinal, 85
    longitudinal, 85
process environment, 86, 88
process model, 86

Racal Research, 24
re-engineering, *70*
Reading Motion Detector, 13, 50
Reading Vehicle Tracker, 94
Real-Time Perspective, 26
red, 32
refactoring, *71*
region, *7*, 8
reverse engineering, 71, *71*
run-length, 8

S-P-E, 87, 90
S-type, 90, 91
saturation, 32, *32*, 41, 42, 44
SGI, 15, 16, 29, 48, 57, 73, 75, 81, 94, 95, 98
Sidenbladh, Hedvig, 21
Société des Transports Intercommunaux Bruxellois, 5
software aging, 91
software artefact, *70*
Software maintenance
    adaptive, 69
    corrective, 69
    perfective, 69
    preventive, 69
SPIN, 87
SSE, 19
static analysis, 76
static information, 71, *71*
Streaming SIMD Extensions,
    see SSE
Sun Microsystems, 73, 94

Testbed 1, 26
Testbed 2, 26
Thales Research, 24
Thales Research Ltd, 5

Use Case, *26*

value, 32, *32*, 41, 42, 44
video matrix switcher, 33
VIEWS, 13
Vigitec, 5, 24

W^4, *16*, 18, 19, 23, 54
W3C, 28
Waterfall model, 90
Word Wide Web Consortium, 28

XML, *28*, 28, 62

XML Schema, 28, *28*, 47, 75
XP, 90

YCC, 32
YIQ, 32
YUV, 32