Shape Knowledge for Segmentation and Tracking



Victor Adrian Prisacariu
Balliol College

D.Phil. Thesis
Supervised by Prof Ian D. Reid

Robotics Research Group
Department of Engineering Science
University of Oxford

Hilary Term 2012

This thesis is submitted to the Department of Engineering Science, University of Oxford, for the degree of Doctor of Philosophy. This thesis is entirely my own work, and, except where otherwise indicated, describes my own research.

Spero melior

Victor Adrian Prisacariu
Balliol College
University of Oxford

Doctor of Philosophy
Hilary Term
2012

Shape Knowledge for Segmentation and Tracking

Abstract

The aim of this thesis is to provide methods for 2D segmentation and 2D/3D tracking, that are both fast and robust to imperfect image information, as caused for example by occlusions, motion blur and cluttered background. We do this by combining high level shape information with simultaneous segmentation and tracking.

We base our work on the assumption that the space of possible 2D object shapes can be either generated by projecting down known rigid 3D shapes or learned from 2D shape examples. We minimise the discrimination between statistical foreground and background appearance models with respect to the parameters governing the shape generative process (the 6 degree-of-freedom 3D pose of the 3D shape or the parameters of the learned space). The foreground region is delineated by the zero level set of a signed distance function, and we define an energy over this region and its immediate background surroundings based on pixel-wise posterior membership probabilities. We obtain the differentials of this energy with respect to the parameters governing shape and conduct searches for the correct shape using standard non-linear minimisation techniques.

This methodology first leads to a novel rigid 3D object tracker. For a known 3D shape, our optimisation here aims to find the 3D pose that leads to the 2D projection that best segments a given image. We extend our approach to track multiple objects from multiple views and propose novel enhancements at the pixel level based on temporal consistency. Finally, owing to the per pixel nature of much of the algorithm, we support our theoretical approach with a real-time GPU based implementation.

We next use our rigid 3D tracker in two applications: (i) a driver assistance system, where the tracker is augmented with 2D traffic sign detections, which, unlike previous work, allows for the relevance of the traffic signs to the driver to be gauged and (ii) a robust, real time 3D hand tracker that uses data from an off-the-shelf accelerometer and articulated pose classification results from a multiclass SVM classifier.

Finally, we explore deformable 2D/3D object tracking. Unlike previous works, we use a non-linear and probabilistic dimensionality reduction, called Gaussian Process Latent Variable Models, to learn spaces of shape. Segmentation becomes a minimisation of an image-driven energy function in the learned space. We can represent both 2D and 3D shapes which we compress with Fourier-based transforms, to keep inference tractable. We extend this method by learning joint shape-parameter spaces, which, novel to the literature, enable simultaneous segmentation and generic parameter recovery. These can describe anything from 3D articulated pose to eye gaze. We also propose two novel extensions to standard GP-LVM: a method to explore the multimodality in the joint space efficiently, by learning a mapping from the latent space to a space that encodes the similarity between shapes and a method for obtaining faster convergence and greater accuracy by use of a hierarchy of latent embeddings.

Acknowledgements

First and foremost I would like to thank my supervisor, Professor Ian Reid for his continuous guidance and essential help and support, without which this thesis would not have been possible.

I would also like to thank Professor Paul Buckley for his permanent encouragements and mentoring and Professor David Murray for his helpful and useful assistance. Furthermore, I also acknowledge the many interesting and insightful conversations I had, while working on this thesis, with present and past members of the Active Vision Lab, such as (in alphabetical order) Ben Benfold, Charles Bibby, Amaury Dame, Siddharth Eswaran, Alex Flint, Olaf Kähler, Carl Yuheng Ren, Alex Segal and Eric Sommerlade.

I also have to thank Lightning Fish Games ltd., Mr. Winston Churchill of the Oxford Mobile Robotics Group and Professor Bodo Rosenhahn from the University of Hannover for allowing me to use their data sets in my experiments.

This work was supported in part by the Engineering and Physical Sciences Research Council and by Balliol College, Oxford.

Last but definitely not least, I would like to thank my parents for all the love and support they have given me throughout the years.

Contents

Contents i

List of Figures iv

List of Tables xvi

1 Introduction 1

1.1 Motivation and Objective . . . 1

1.2 Approach and Thesis Outline . . . 3

1.3 Contributions . . . 5

1.4 Publications . . . 7

2 Literature Review 10

2.1 Introduction . . . 10

2.2 Model-Based Segmentation and Tracking . . . 11

2.2.1 Edge Based Methods . . . 12

2.2.2 Point-Based Methods . . . 15

2.2.3 Region Based Methods . . . 17

2.3 Shape Space-Based Segmentation and Tracking . . . 20

2.3.1 Dimensionality Reduction . . . 21

2.3.2 Shape Representation . . . 27

2.3.3 GP-LVM for Articulated Pose Tracking . . . 32

2.4 Conclusions . . . 33

3 PWP3D: Real-time Segmentation and Tracking of 3D Objects 35

3.1 Introduction . . . 35

3.2 Notation . . . 37

3.3 Segmentation and Pose Tracking . . . 38

3.3.1 Optimisation Algorithm . . . 41

3.3.2 Multiple Objects . . . 44
3.3.3 Multiple Views . . . 45
3.3.4 Invisible Points . . . 45
3.3.5 Temporal Consistency . . . 47
3.3.6 Online Adaptation . . . 48
3.4 Implementation and Performance Analysis . . . 48
3.5 Results . . . 54
3.6 Conclusions . . . 62
4 PWP3D Applications **67**
4.1 Introduction . . . 67
4.2 Driver Assistance . . . 68
4.2.1 Overview . . . 70
4.2.2 Object Detection and Recognition . . . 71
4.2.3 Non-Maximum Suppression . . . 71
4.2.4 Approximate 3D Pose Reconstruction and 3D Tracking . . . 73
4.2.5 Results . . . 75
4.3 Human Computer Interaction . . . 79
4.3.1 Overview . . . 81
4.3.2 Visual 3D Tracker . . . 82
4.3.3 Accelerometer . . . 83
4.3.4 Pose Classifier . . . 88
4.3.5 Descriptor . . . 91
4.3.6 Multiclass Classifier . . . 91
4.3.7 Results . . . 92
4.3.8 Quantitative Benchmark . . . 96
4.4 Conclusions . . . 103
5 Segmentation and Tracking with Shared Shape Spaces **105**
5.1 Introduction . . . 105
5.2 Learning Shape Spaces . . . 110
5.2.1 Hierarchy of Manifolds . . . 113

5.2.2 Manifold Modes Discovery . . . 115
5.2.3 Shape Representation . . . 117
5.3 Inference . . . 122
5.3.1 Shape Minimisation . . . 123
5.3.2 Pose Minimisation . . . 127
5.3.3 Parameter Recovery . . . 129
5.4 Results . . . 129
5.4.1 Segmentation . . . 130
5.4.2 Pose Recovery . . . 136
5.4.3 Parameter Recovery . . . 139
5.4.4 Failure Cases . . . 143
5.5 Conclusions . . . 144
6 Conclusions **146**
6.1 Summary . . . 146
6.2 Future Work . . . 148
6.2.1 Low Level Improvements . . . 149
6.2.2 High Level Improvements . . . 150
Appendix **153**
A.1 fastHOG . . . 153
A.1.1 HOG Descriptor . . . 153
A.1.2 Descriptor Computation . . . 155
A.1.3 Linear SVM Evaluation . . . 158
A.1.4 Benchmarks . . . 159
Bibliography **161**

List of Figures

1.1 Example 3D pose convergence using the algorithm presented in Chapter 3. 6

1.2 Example human-computer interface based on the algorithm shown in Chapter 4. 7

1.3 Example 3D shape and 3D pose using the algorithm presented in Chapter 5. 8

1.4 Example joint latent space (left) between 3D articulated poses and 2D silhouettes for a walk, obtained using the algorithm presented in Chapter 5. 8

2.1 RAPiD control points and distances from projected to image edges. . 13

2.2 Example results from Drummond and Cipolla (2002). Image taken from Drummond and Cipolla (2002), ©2002 IEEE. 14

2.3 Example results from Skrypnyk and Lowe (2004). Image taken from Skrypnyk and Lowe (2004), ©2004 IEEE. 16

2.4 Example level set function embedding the contour of the projection of a soft-drink can 3D model. The size of the image containing the contour is 640×480. 18

2.5 Example results from Gall et al. (2008). Images taken from Gall et al. (2008), ©2008 IEEE. 19

2.6 Example mean curve (middle) and curves corresponding to plus (above) and minus (below) one standard deviation in the parameters. Image taken from Staib and Duncan (1992), ©1992 IEEE. 28

2.7 Example shape variability for a fighter jet. (a) – mean image; (b) $+1\sigma$ (left) and -1σ (right) variability on the first mode of variation; (c) $+1\sigma$ (left) and -1σ (right) variability on the second mode of variation; (d) $+1\sigma$ (left) and -1σ (right) variability on the third mode of variation. Images taken from Tsai et al. (2003), ©2003 IEEE. 30

2.8 Example segmentation for a fighter jet. (a) – original image; (b) – original image with line occlusions and Gaussian noise; (c) – initialisation contour; (d) final result. Images taken from Tsai et al. (2003), ©2003 IEEE. 30

2.9 Improvements brought by the use of KPCA over PCA. (a) – initial image; (b) – results when using PCA; (c) – result using KPCA. The KPCA results is much more realistic than the PCA result. Image taken from Dambreville et al. (2008a), ©2008 IEEE. 31

3.1 Representation of the object showing: the contour of the projection $\mathbf{C}$ and the corresponding contour around the visible part of the 3D model, the foreground region Ω_f and the background region Ω_b, a 2D point $\mathbf{x}$ on the contour and its corresponding 3D point in the object coordinate frame $\mathbf{X}$. 37

3.2 Setup (image, model used and initialisations) for the convergence experiment. 44

3.3 Importance of considering the most distant points. As the camera moves from (a) to (b) the evolution of the silhouette is not dictated by the blue points (closest points) but rather by the green points, which were the most distant points in (a). 46

3.4 The average processing time when using the model on the right with the image on the left is 2.91ms. With this model the algorithm usually required between 15 and 25 iterations for convergence, resulting in an average processing speed of 15-20 frames per second. The resolution of the images is 640×480 and the model has 680 triangles. 49

3.5 3D Rendering Pipeline 50
3.6 Typical algorithm run for one frame. left - initialisation, middle - intermediate iteration, right - final result. 54
3.7 Failure due to the symmetry of the object. (a) – left: recovered pose, right: rendered frame, (b) – left: recovered pose, right: rendered frame. 55
3.8 Robustness to motion blur. Left – original image, Right – recovered position. 56
3.9 Effect of temporal consistency on the posterior probabilities. The top row shows the original images. In the second and third row, we plot the per pixel difference $P_f - P_b$ where $P_f - P_b > 0$, scaled to $[0, 255]$. The separation between the foreground and background regions is better when imposing temporal consistency (third row). 57
3.10 Film strip showing our method tracking a hand. Our algorithm can track the hand in this sequence at around 20fps. 57
3.11 Film strip showing 5 frames from a video sequence in which the camera moves above a soft-drink can in a circular motion. Between frames 25 and 55 the view of the can moves to directly above, with the side obscured, back to a view with the side visible again. When allowing only the closest points to influence the contour evolution, the pose incorrectly flips (above). In contrast the behaviour is correct when the furthest points are also considered (below). 58
3.12 Performance analysis: Probability of convergence to the correct pose generated from over 200,000 experiments. 59

3.13 Film strip consisting of 10 frames from a video which showcases the PWP3D behaviour when subjected to occlusions and high levels of noise. For the first 6 frames (when both the top and bottom parts of the soft-drink can are visible) the pose is correct, even though most of the interior part of the object is occluded. When the bottom part of the can becomes occluded the pose becomes incorrect. The object moves "up" in the image because there is no longer a reason for it to stay "down". The algorithm converges back to the correct position once the bottom part stops being occluded. 60

3.14 Comparison between our method (above) the one presented by Brox et al. (2009). Although Brox et al. (2009) combine region-based tracking with optical flow and SIFT features to achieve tracking, while we use only the silhouette, our results are very similar, if not better. Note that here we use a single view (above, first row), and reproject in the other (above, second row), while Brox et al. (2009) use three views (from which we included 2 here). 61

3.15 Filmstrip showing 6 frames from our PTAM comparison video. We used a checker board pattern to calibrate our results with the PTAM (Klein and Murray, 2007) ones. 63

3.16 Comparison between camera pose output when using PTAM (Klein and Murray, 2007) and PWP3D. 63

3.17 Multi view tracking. The top two rows show the two views. When processing only the first view (top row) the results are incorrect (third row) because the silhouette to pose mapping is ambiguous (multiple silhouettes can be generated by the same pose). When using two views the ambiguities are eliminated and the results are correct for both views (bottom two rows). 64

3.18 Our algorithm tracking two objects, a soft drink can and a hand. The top film strip shows the hand occluding the soft drink can. The bottom film strip shows the opposite, the soft drink can occluding the hand. The third row in each film strip shows the resulting depth map. 65

4.1 Importance of determining the traffic sign orientation. The no right turn sign (located in the upper left part of the image) does not point towards the car. 69
4.2 Algorithm Overview 70
4.3 Non-maximum Suppression Example Results 72
4.4 3D–2D point correspondences 73
4.5 System performance while tracking a circle shaped sign over 60m, with just the 4 point pose recovery (RPP) and with the tracker (PWP). . . 76
4.6 System performance while tracking a rectangle shaped sign over 60m, with just the 4 point pose recovery (RPP) and with the tracker (PWP). 77
4.7 System performance while tracking a triangle shaped sign over 60m, with just the 4 point pose recovery (RPP) and with the tracker (PWP). 77
4.8 Filmstrip showing frames from a video tracking a pedestrian crossing sign. 78
4.9 Filmstrip showing frames from a video tracking a triangular sign. . . . 78
4.10 Filmstrip showing frames from a video tracking multiple round signs. . 78
4.11 Filmstrip showing frames from a video tracking a parking sign, in spite of it being occluded by a tree. 78
4.12 System Overview 82
4.13 3D Models. (a) is used in Section 4.3.8; (b),(c),(d) are used in Section 4.3.4. 82
4.14 Silhouette - Pose ambiguities: (a) real example; (b) artificial example. 83
4.15 Hand with accelerometer sensor board 83
4.16 Rotation quaternions and coordinate systems for the visual 3D tracker and the accelerometer 84

4.17 Example calibration motion 85
4.18 Charts comparing our calibration method to Lang and Pinz (2005). We use two known 500 frame motions, q_1 and q_2, with a known calibration quaternion. We first add a small amount of noise at each frame (random angle in [-5°, 5°]) and compute the calibration quaternion, using the two methods. Both recover the calibration quaternion successfully. Next, we keep this source of noise and add another, bigger, one every 60 frames (random angle in [-30°, 30°]). Our method again recovers the correct calibration quaternion, while the method of Lang and Pinz (2005) fails. 87
4.19 Pose classification overview 89
4.20 Example coarse hand segmentations obtained by the PWP algorithm. 90
4.21 Example final segmentation results. 90
4.22 Example training data: (a) – class 1; (b) – class 2; (c) – class 3; (d), (e) – negative class. The models for (a), (b) and (c) are shown in Figure 4.13. 91
4.23 Filmstrip showcasing robustness to motion blur. The top row shows the original camera image, the middle row shows a zoomed snapshot of the hand and the bottom row shows the result obtained by our algorithm. 92
4.24 Robustness to occlusions: top filmstrip – as the hand moves behind the soft drink can, the colour models have not yet adapted and the results are incorrect; bottom filmstrip – after the histogram has adapted the results correct. 93
4.25 Filmstrip showing tracking results with and without the accelerometer. The top row shows the original camera image. The middle and bottom rows show OpenGL renderings of the hand in the recovered pose, without and with the use of the accelerometer, respectively. The visual 3D tracker alone cannot deal with ambiguities in the silhouette (where a change in pose does not alter the silhouette). The accelerometer provides enough information to discriminate between the ambiguous poses. 94

4.26 Filmstrip showing tracking results with and without the accelerometer. The top row shows the posterior map (the per pixel difference $P_f - P_b$, where $P_f - P_b > 0$). The middle and bottom rows show wireframe renderings made by our rendering engine and used by the visual tracker. The maximum allowed number of iterations was not enough for the visual tracker alone to converge, which quickly finds an incorrect local minima. The accelerometer positions the visual tracker closer to the correct result, so fewer iterations are required and the results are better. 95

4.27 Filmstrip showing tracking a typical failure case. The accelerometer does not provide any yaw, so it cannot help the visual tracker disambiguate between similar silhouettes. 96

4.28 Our system tracking multiple hand configurations and proof-of-concept augmented reality application. 97

4.29 Comparison between our method and the ground truth, focusing on translation. 98

4.30 Comparison between our method and the ground truth, focusing on rotation. 99

4.31 Example frames and results from the translation accuracy test. Original image (first row), wireframe (second row) and full rendering (third row) results for our method, with and without the accelerometer. 100

4.32 Example frames and results from the rotation accuracy test. Original image (first row), wireframe (second row) and full rendering (third row) results of our method, without the accelerometer, wireframe (forth row) and full rendering (fifth row) results of our method, with the accelerometer. 101

4.33 Example human-computer interface based on our tracker. Top filmstrip – calibration – the user marks a plane by moving the hand along the edges of the board. Middle filmstrip – the user places a DVD box on the surface, prompting the system to show facts about the DVD box and a 3D object related to it, which can be manipulated with the hand. For the first DVD, the user can interact with a 3D model of a starship. For the second DVD, the user can interact with a model of planet Mars. Bottom filmstrip – the user can draw on the 2D surface, transform the 2D drawing into a 3D object (by tapping inside it and lifting the hand) and pick up and move the 3D object. 103

5.1 Shared GP-LVM graphical model. $\mathbf{Y}$ and $\mathbf{Z}$ are the observation spaces (the shape and the parameter spaces in our case) and $\mathbf{X}_{YZ}$ is the lower dimensional joint latent space. 111

5.2 Graphical model for our hierarchy, when using two observation spaces. $\mathbf{Y}$ is the shape space, $\mathbf{Z}$ the parameter space and $[\mathbf{X}^0_{YZ}, \ldots, \mathbf{X}^r_{YZ}]$ the hierarchical shared shape-parameter spaces. 113

5.3 Example similarity function 115

5.4 Graphical model for our modes discovery stage. $\mathbf{Y}$ is the shape space, $\mathbf{S}$ the similarity space, $\mathbf{X}_{YS}$ the shared space between shape and shape similarity and $\mathbf{X}_{YZ}$ the shaped space between shapes and sets of parameters. 116

5.5 Example of our method discovering all the modes in the silhouette space of a person rotating by 360 degrees on a single axis. $\mathbf{X}_{YS}$ is the shared shape–similarity function space and $\mathbf{X}_{YZ}$ is the shared shape–pose space. Warmer colours (e.g. red) indicate higher variance, while colder colours (e.g. blue) indicate lower variance. The point marked in $\mathbf{X}_{YS}$ generates a shape $\mathbf{y}$ and a function $s_{\mathbf{y}}$ (3rd and 4th rows). The top three peaks of $s_{\mathbf{y}}$ indicate the top three $\mathbf{x}_{YZ}$ points that are likely to generate similar shapes (2nd row). The last three rows show the back-projections of the peaks to both shape and pose spaces. 117

5.6 Example 3 harmonics of an elliptic Fourier descriptor. 118

5.7 Distance transform accuracy example. The contour is the zero level of the middle distance transform, which is generated from a 2D latent space. The chart shows vertical and horizontal cross sections in the per pixel error between this distance transform and the ground truth one, for the same contour. 121

5.8 Example 3D model and projection: left – projection, blue represents $p_{fg} = 0$, red represents $p_{fg} = 1$; right – the 3D model that generated the projection. 122

5.9 2D latent space capturing the 2D shape kinematics of a person performing a star jump (left) together with samples from the training data (right, top row) and the latent space (right, bottom two rows). Warmer colours (e.g. red) indicate higher variance, while colder colours (e.g. blue) indicate lower variance. Samples 1–9 are taken from a low variance region (making them more likely to generate valid shapes) while 10 and 11 are from taken from higher variance regions (making them less likely to generate valid shapes). 131

5.10 Example convergence of our method: left – values of the function for meaningful points in the latent space; middle – samples from the convergence trajectory; right – result (surrounded with blue) overlayed with initialisation (transparent black). Minimisation starts at the point marked with s and ends at the point marked with t. 131
5.11 Impact of latent space dimensionality. A 4D latent space (b) leads to a better segmentation than a 2D latent space (a). 132
5.12 Example 2D latent space of side view car silhouettes (left), together with training data examples (right, top row) and latent space samples (right, bottom three rows). Samples 1-12 are taken from a low variance region of the space (more likely valid shapes) while samples 13 and 14 are taken from a high variance region (less likely valid shapes). 132
5.13 Example segmentations of cars: (a) – original images; (b) – map of $P_f - P_b$, where $P_f > P_b$; (c) – result using Grab Cuts (Rother et al., 2004); (d) – result using PWP (Bibby and Reid, 2008); (e) – our result. 133
5.14 Examples of our algorithm dealing with occlusions. 133
5.15 Accuracy comparison between our method using an implicit shape representation (red, circle marking), our method using an explicit shape representation (green, x marking), and the standard ones of using Kernel PCA (Dambreville et al., 2008a) (blue, dot marking) and PCA (Tsai et al., 2003) (black, triangle marking). 134
5.16 Tracking example. The latent space segmentation is combined with a 2D pose recovery step, with each minimisation being alternately iterated to convergence. The top row shows the latent space and convergence trajectory for each example frame. The starting shape is shown in black. The convergence trajectory starts from the s point and ends with the t point. The bottom row depicts the result. 136
5.17 Example convergence steps for the 2D pose / latent space one shot minimisation. 137

5.18 Example 2D latent space of 3D car shapes. Shapes 1-7 are points inside the low variance region of the space (in blue), while shape 8 is from the high variance area (in red). The sample from the high variance area does not have a corresponding ground truth because it was not part of the original training data. 137
5.19 Example 3D shape and 3D pose convergence for our algorithm. 138
5.20 Film strips showing our algorithm successfully tracking different types of cars in different environments. 138
5.21 Charts showing recovered latent space positions, with (in red) and without (in blue) shape consistency. When shape consistency is used the trajectories in the recovered latent space are smoother and tighter. . . 139
5.22 Failure due to occlusion combined with lack of shape consistency. When the cyclist passes in front of the camera and shape consistency is not kept, the system fails (top two rows). When information from multiple frames is integrated the system successfully tracks through the occlusions (bottom two rows). 140
5.23 Example joint latent space (left) for side view car silhouettes and 3D models. Samples 1-6 are taken from a low variance region of the space (so more likely valid shapes) while sample 7 is taken from a high variance region (so less likely valid shapes). 140
5.24 Examples of our algorithm correctly generating 3D car models from 2D car silhouettes. 140
5.25 Gaze tracking. The latent space (middle column, top) generates both eye sclera shapes and location coordinates. To test and train we displayed a grid of dots (spaced at 32.23mm) on the screen (middle column, bottom) and looked towards each one. Images 5-12 show training shapes. Their respective screen locations are marked green on the grid. Images 1-4 show segmentation results. Positions 1-4 on the grid show the ground truth (blue) and the value generated by our algorithm (red). 141

5.26 Example joint latent space (left) between 3D poses and 3D silhouettes for a walk from the HumanEva data set. 142
5.27 Example HumanEva results (4 accurate and 2 failure cases). Dotted line marks ground truth and filled line marks our results. 142
5.28 Failure cases: left – failure due to lack of training data; right – failure due to high degree of occlusion. 144

6.1 Comparison between the foreground / background separation provided by the PWP method and a RF classifier. Warm colours indicate high probability and cold ones low probability. 150
A.1.1 Example results after applying the HOG detector for pedestrians (left) and heads (right) 154
A.1.2 fastHOG steps for computing the HOG descriptor 155
A.1.3 Thread configuration for the block histogram step. The dark area represents one block, split into cells. The vertical arrows represent threads and the direction they process the pixels. Each thread processes one column from once cell in a block. 157
A.1.4 A 2×2 block and the memory write pattern used for trilinear interpolation of the histogram bins. Each cell is split into 4 subcells. The pixels in the subcell marked with 1 contribute to the histogram of cell 1, the pixels in the subcell marked with 1 and 3 contribute to the histogram of cells 1 and 3, etc. 157
A.1.5 Thread configuration for the SVM evaluation step. The vertical arrows represent threads, and the direction they process the pixels. For our case, where a window has 64×128 pixels equal to 7×15 blocks, a thread block has 7×18 threads, each thread processing 15×2 values. 159

List of Tables

3.1 Partial differentials of the 3D point in the camera frame $\mathbf{X}$ with respect to the pose parameters. 40

3.2 Number of iterations and timing details required for convergence when using gradient descent with (*GD*) and without (*GD-LS*) a line search step and conjugate gradients with (*CG*) and without (*CG-LS*) a line search step. The time per iteration for translation is smaller than that for rotation because computing the derivatives with respect to rotation requires considerably more mathematical operations (about 10 times more). The setup for this experiment is depicted in Figure 3.2. 43

3.3 Detailed processing time for one iteration of the minimisation of our energy function, when using gradient descent without the line search (so no evaluation of the energy function is necessary). 50

5.1 Average convergence accuracy, with and without the hierarchy. 136

5.2 Average convergence speed (seconds) per frame, with and without the hierarchy. 136

5.3 Performance of our algorithm on the Poser data set (Agarwal and Triggs, 2004) 143

A.1.1 Processing time for the INRIA Person test set 160

Chapter 1

Introduction

This chapter details the motivation behind this thesis, the objectives, major contributions and basic outline of our work.

1.1 Motivation and Objective

Segmentation is the process of separating an image into multiple group of pixels, one for the background and one for each object contained in that image. It usually follows a typical pipeline: first, some form of features (edges, colour, texture, etc.) are extracted from the image. Next, these features are distilled into image statistics and an optimisation is used to find the boundaries separating the object(s) from the background. Often linked to segmentation, tracking is the process of localising each object in each frame from a video sequence. A typical pipeline consists of first segmenting the object from the background, then selecting features from within its boundaries and finally searching for these features across all available frames.

The two tasks are fundamental to imaging and vision and are vital to many applications. For example, the first step in many human computer interaction systems is to segment and track the various human body parts. Another example is the diagnosis of heart disease, which often requires the heart to be first segmented from MRI or ultrasound data. Yet another example is surveillance, where humans need to

be tracked throughout multiple video streams coming from multiple video cameras.

Most such real world scenarios require robustness to noise, occlusions, imperfect illumination, motion blur, etc. and fast, ideally real time, performance. In spite of several decades of research, these requirements still pose considerable challenges.

Lack of robustness is due, we believe, at least in part, to two causes.

First, much of the current state of the art in segmentation and tracking is based on local image features (local edges, local pixel colour, etc.). On the other hand, the human visual system has been proven, in recent psychology and neurobiology studies such as Kersten et al. (2004) and Kourtzi and Kanwisher (2001), to leverage global information. This affords it a much higher resilience to image ambiguity. The human brain uses prior information on object shape, material and lighting. This prior information could be acquired, among other means, genetically or via opportunistic learning (i.e. the brain learns the shape of the object when the ambiguity of the observation is low) (Brady and Kersten, 2003). These priors are not local and feature specific, but rather global, very abstract and high level, as is shown by Kourtzi and Kanwisher (2001).

Second, a common trademark of the current state of the art is the dissociation between segmentation and tracking. Most often, objects are first segmented and then tracked, which leads to problems in both tasks. On the one hand, imperfections in segmentation lead to failure in tracking. On the other hand, frame-to-frame variability in the apparent shape of the object, that should be explained by a change in position, is actually being explained by a change in its shape. Formally, this often translates into a very low dimensional variability in pose being explored using, effectively, infinite dimensional shape searches.

While robustness may be sought, speed is unfortunately often ignored. It is too common that tracking or segmentation approaches are developed that take tens of minutes or hours per frame, which prohibits their use in many real world applications.

The objective of this thesis is to provide fast methods for 2D segmentation and 2D/3D tracking that are robust to imperfect image information, as caused by occlusions, motion blur, uneven illumination, noise or cluttered backgrounds. This aim in

itself is not novel, as several attempts have been made in the past. These are however, at best, impractical and, at worst, non-functioning. We aim to do this, by developing **fast**, ideally **real time**, efficient, generic and principled ways of using **high level** shape knowledge in **simultaneous** segmentation and tracking.

1.2 Approach and Thesis Outline

The work presented in this thesis is based on the assumption that the space of possible 2D shapes an object can take in an image can be either (i) generated by projecting the known rigid 3D shape of the object, or (ii) learned from other 2D shape examples of the same type of object. The "type" of object can refer either to its class (e.g. all types of cars, all types of planes, etc.) or to the activity it is doing (e.g. running, walking, etc). Previous work on both these methodologies is detailed in Chapter 2.

We treat both methodologies in a similar fashion and cast the segmentation process as a search inside either the generated or the learned shape space. We do this by minimising an image-based segmentation quality measure either (i) with respect to the 6 degree-of-freedom (DoF) pose parameters of the known 3D object shape or (ii) with respect to the parameters of the learned space. In both cases this is done by computing the derivatives of the energy function, with respect to the pose parameters or the parameters of the learned space, and using standard nonlinear minimisation algorithms (gradient descent or conjugate gradients). Using a fixed 3D model and optimising for 3D pose has the obvious advantage that, along with 2D segmentation, we also obtain 3D rigid pose. This does however restrict us to rigid 3D shapes. Learning a space of shapes does not pose this limitation, and it can be extended to provide, along with 2D segmentation, generic types of parameters, including, for example, 3D pose, 3D shape or gaze. It is however considerably slower. Whilst the first method runs in real time, the second one requires several seconds per frame. We have also combined the two approaches, by learning spaces of 3D shapes, and iterating both pose and shape.

Throughout this thesis we represent 2D shapes implicitly as the zero levels of

level set embedding functions. This allows us to work with all types of 2D shapes (composed of single or multiple pieces, with or without holes, etc.). Also throughout the whole thesis, for the segmentation quality measure, we draw inspiration from the level set and region based energy function of Bibby and Reid (2008). This assumes that an image can be split into a foreground and a background region and the statistics of these regions (or approximations thereof) are known in advance. Bibby and Reid (2008) adopt the use of pixel wise posteriors (as opposed to likelihoods) in their work, and show that, in the context of 2D segmentation and tracking, their energy function results in a larger convergence basin with fewer local minima. In this thesis, we show that these benefits extend to a 3D context.

In Chapter 3 we explore rigid 3D object tracking. We search for the 3D six degree-of-freedom pose, which projects a known 3D model down to an explicitly defined contour (using a standard projection pipeline), which, embedded inside a level set function, minimises the Bibby and Reid (2008) energy function. We represent 3D shapes explicitly via standard CAD models (sets of 3D vertices). The parallel nature of much of the processing in our algorithm means it is amenable to GPU acceleration, and we give details of our real-time implementation. As the per pixel probability of foreground or background membership should not change instantly, we also impose per pixel temporal consistency. Two applications of this method are detailed in Chapter 4. In the first one we link our 3D pose recovery algorithm to a 2D detection phase, to create a real time driver assistance system. In the second one we augment our 3D pose recovery method with 3D rotation coming from an inertial sensor, which we place on the hand, to create a 3D hand tracking system. We also introduce limited articulated tracking, by training a classifier to separate between a small number of hand configurations.

In Chapter 5 we explore deformable 3D object tracking. As before, we only use 2D shape to provide image information. This makes any attempt at an unconstrained optimisation of 3D shape and 3D pose very likely to fail. A key challenge here therefore is finding a way to effectively restrict the pose/shape search space. We do this by learning low dimensional latent spaces of shapes using a dimensionality reduction

technique called Gaussian Process Latent Variable Models (GP-LVM) (Lawrence, 2005). We can represent both 2D and 3D shapes, either implicitly or explicitly and keep inference tractable by compressing them with Fourier-based transforms (elliptic Fourier descriptors for explicit shapes and discrete cosine transforms for implicit shapes). Whereas the approach from Chapter 3 can generate 2D segmentation and 3D pose, the method presented in Chapter 5 enables us to learn shared GP-LVM shape – parameter spaces, which generate both 2D segmentation and generic sets of parameters. These parameters can describe anything from 3D articulated pose to 3D shape or eye gaze. We also propose a method for exploring the inherent multimodality in the joint space in an efficient manner, by learning a mapping from the latent space to a space that encodes the similarity between shapes. We further extend GP-LVM to a model that makes use of a hierarchy of embeddings and show that this yields faster convergence and greater accuracy over the standard non-hierarchical embedding.

In Chapter 6 we present a thesis summary, the conclusions that can be drawn from our work and possible avenues for future research.

1.3 Contributions

We have made a number of contributions to the overall areas of 2D segmentation and 3D tracking. Below we summarise them and indicate the chapter that details them and where they have been published in the peer reviewed literature.

Chapter 3 and (Prisacariu and Reid, 2009b, 2012b): a model based 3D tracker that is region and level set based and has a variational and probabilistic formulation. Being region based, it is robust to clutter and can work in real world environments. No presegmentation is required, meaning it is robust to occlusions, motion blur and noise. We also show how it extends to multiple objects and views, and how the pixel-wise image statistics can be improved by imposing temporal consistency. Unlike previous work, our algorithm is suited for very high level parallelisation, so, using a GPU based implementation, we obtain real time performance. Figure 1.1 shows an example convergence for this 3D tracker.



Figure 1.1: Example 3D pose convergence using the algorithm presented in Chapter 3.

Chapter 4 and (Prisacariu et al., 2010; Timofte et al., 2011): a real-time driver assistance system that integrates single view detection with region-based 3D tracking of road signs. Unlike previous work, our system allows us to obtain the 3D orientation of each road sign, for any type of traffic sign, from a single camera placed on a car. This is joint work with Radu Timofte from the Katholieke Universiteit Leuven, Belgium.

Chapter 4 and (Prisacariu and Reid, 2009a, 2011b, 2012a): a real-time model-based 3D hand tracker, that combines image regions and the signal from an off-the-shelf 3-axis accelerometer placed on the user's hand. The visual regions allow the tracker to cope with occlusions, motion blur and background clutter, while the latter aids with the inherent silhouette-pose ambiguities. The accelerometer and tracker are synchronised by casting the calibration problem as one of principal component analysis. Based on the assumption that, often, the number of possible hand configurations is limited by the activity the hand is engaging in, we use a multiclass pose classifier to distinguish between a number of activity dependent articulated hand configurations. Figure 1.2 shows an example human computer interaction based on our hand tracking algorithm.

Chapter 5 and (Prisacariu and Reid, 2011a,c): a method for simultaneous shape-constrained segmentation and parameter recovery. The parameters can describe anything from 3D shape to 3D pose and we place no restriction on the topology of the shapes, i.e. they can have holes or be made of multiple parts. Shapes can be 2D or 3D and are represented either explicitly or implicitly (using level sets), and inference is made tractable by compressing them with Fourier based transforms. We

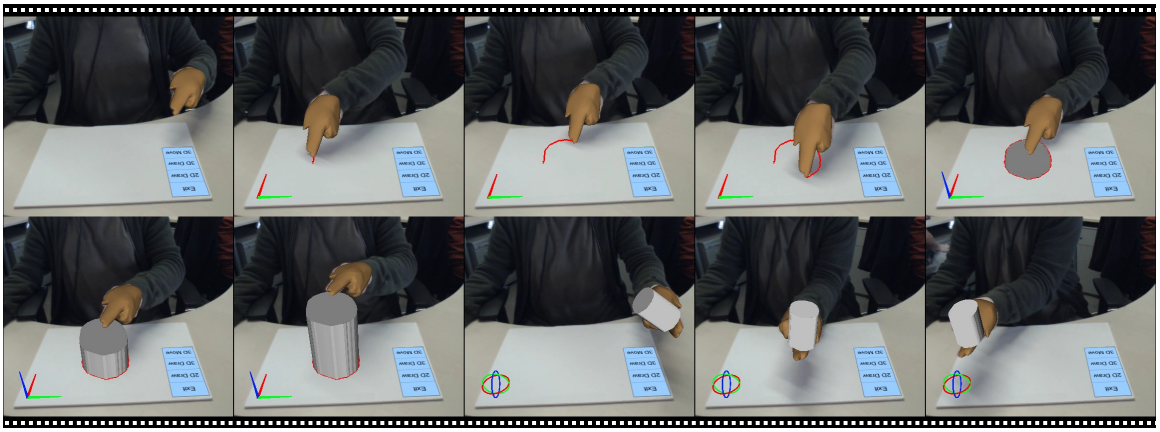


Figure 1.2: Example human-computer interface based on the algorithm shown in Chapter 4.

use (Shared) Gaussian Process Latent Variable Models to learn multimodal shape-parameter spaces. These allow for non-linear embeddings of the high-dimensional shape and parameter spaces in low dimensional spaces in a fully probabilistic manner. We further propose a method for exploring the multimodality in the joint space in an efficient manner, by learning a mapping from the latent space to a space that encodes the similarity between shapes. We also further extend SGP-LVM to a model that makes use of a hierarchy of embeddings and show that this yields faster convergence and greater accuracy over the standard non-hierarchical embedding. We show state of the art results in various fields, ranging from pose recovery to gaze tracking and to monocular 3D reconstruction. Figure 1.3 shows an example convergence for our joint 3D shape and 3D pose optimisation and Figure 1.4 shows an example 2D latent space capturing 3D articulated pose and 2D shape obtained for a walk sequence.

1.4 Publications

The work detailed in this thesis has been published in various conferences and journals:

- Victor Adrian Prisacariu, Ian Reid - PWP3D: Real-time segmentation and tracking of 3D objects - the International Journal of Computer Vision.

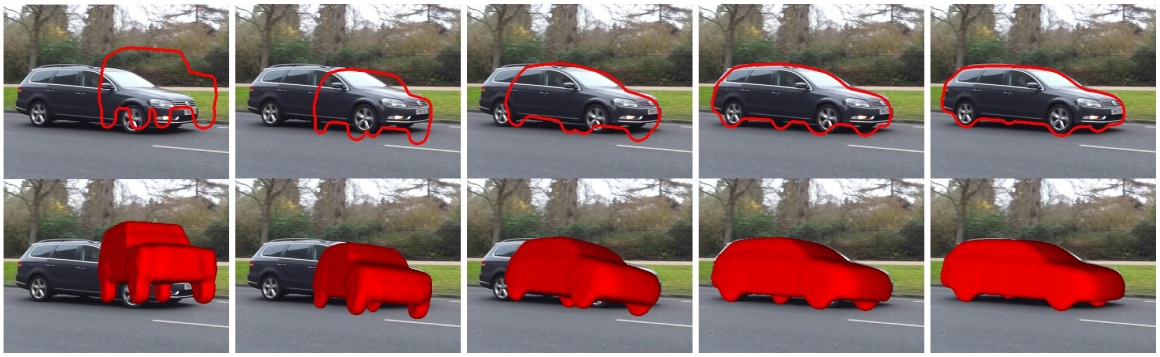


Figure 1.3: Example 3D shape and 3D pose using the algorithm presented in Chapter 5.

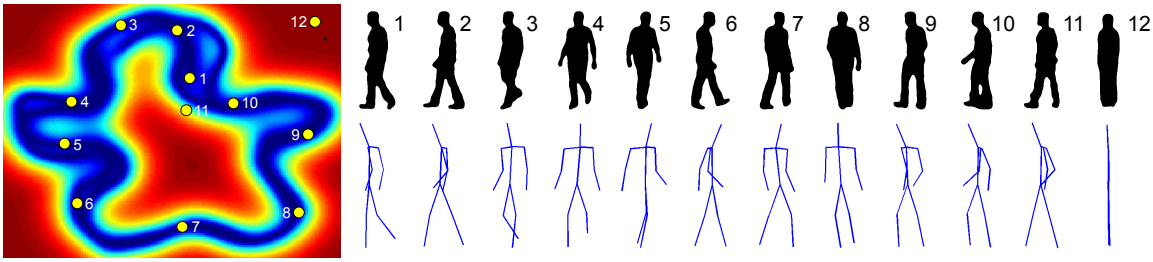


Figure 1.4: Example joint latent space (left) between 3D articulated poses and 2D silhouettes for a walk, obtained using the algorithm presented in Chapter 5.

- Victor Adrian Prisacariu, Ian Reid - 3D Hand Tracking for Human Computer Interaction - the Image and Vision Computing journal.
- Victor Adrian Prisacariu, Ian Reid - Shared Shape Spaces - the 13th International Conference of Computer Vision, Barcelona, 2011.
- Victor Adrian Prisacariu, Ian Reid - Nonlinear Shape Manifolds as Shape Priors in Level Set Segmentation and Tracking - the IEEE Computer Vision and Pattern Recognition Conference, Colorado Springs, 2011.
- Victor Adrian Prisacariu, Ian Reid - Robust 3D Hand Tracking for Human Computer Interaction - the 9th IEEE International Conference on Automatic Face and Gesture Recognition, Santa Clara, 2011.
- Victor Adrian Prisacariu, Ian Reid - PWP3D: Real-time segmentation and tracking of 3D objects - the 20th British Machine Vision Conference, London, 2009.

- Victor Adrian Prisacariu, Radu Timofte, Karen Zimmermann, Ian Reid, Luc Van Gool - Integrating Object Detection with 3D Tracking Towards a Better Driver Assistance System - the 20th International Conference on Pattern Recognition, Istanbul, 2010.

- Radu Timofte, Victor Adrian Prisacariu, Luc Van Gool, Ian Reid - Combining Traffic Sign Detection with 3D Tracking Towards Better Driver Assistance - book chapter in *Emerging Topics on Computer Vision and its Applications*.

- Victor Adrian Prisacariu, Ian Reid - fastHOG - a real-time GPU implementation of HOG - Technical Report 2310/09 Department of Engineering Science, Oxford University.

Chapter 2

Literature Review

This chapter introduces the main techniques (of using shape information in segmentation and tracking) upon which this thesis is based. We search for the advantages and the disadvantages of each method, discuss the current state of the art and set the scene for the remainder of the thesis.

2.1 Introduction

There exist many schools of thought on how shape knowledge is to be represented and used in segmentation and tracking. Here we primarily focus on the two that are most related to our research: the one that aims to generate 2D shapes as projections of known 3D models, and the one that attempts to learn the spaces of possible shapes from examples.

The first class of methods considers the segmentation of an object as a function of the 3D pose of its known rigid 3D model. Finding it is usually formulated as a 6 degree-of-freedom (DoF) optimisation with respect to the rigid pose of this model. The main advantage of such methods is that, along with the segmentation, they also produce the 3D pose of the object. Shape representation tends to be compact and performance can be real-time. However, since the search space increases greatly with the number of DoFs, it is difficult to model objects whose shapes are defined by more

than just a 6 DoF pose (i.e. articulated or deformable object).

The second class of methods learns (linear or nonlinear) shape spaces. The final segmentation result is then constrained (lightly or strongly) to the shapes generated from the learned space. In some cases, the segmentation optimisation itself can be constrained to the space of shapes. Such methods can model complex deformations, but are limited to generating only segmentation (i.e. they do not generate pose). They also tend to be slow and require very large amounts of memory.

In this chapter we review the current state-of-the-art in both classes of methods. We begin in Section 2.2 with an overview of model-based works. These have been traditionally seen as pose tracking methods, a full review of which has been presented by Lepetit and Fua (2005). Here we only focus on methods which can produce 2D segmentations of real objects, i.e. that do not require preliminary segmentation steps and use full (ideally arbitrary) 3D models. Section 2.3 focuses on the other class of methods, namely shape space-based works. We conclude in Section 2.4.

2.2 Model-Based Segmentation and Tracking

A variety of features has been used for 3D model based tracking, including edges, point features and regions. By far the most popular methods to date have been edge-based, where the aim is to find the pose in which the projection of model edges aligns best with edges found in an image. This is perhaps because of the natural viewpoint invariance afforded by the edges and because the extent of edges in an image means that correspondence search can be done in 1D, rather than over two dimensions of the image as for point features. We detail such methods in Section 2.2.1. These systems are all restricted to polyhedral and simple surface models. Later approaches called interest point-based methods can model more general shapes and typically proceed by first establishing correspondences between projected point locations and image points. These are explored in Section 2.2.2. Finally, the most recent approaches, region based methods, work by aligning the contour of the projection of an arbitrarily shaped 3D model with a regional segmentation of the image. This results in robustness to

occlusions, clutter and motion blur, but also in slowness and clumsiness. A number of such methods are detailed in Section 2.2.3.

2.2.1 Edge Based Methods

Most early work in model-based 3D tracking was edge based. Two early seminal works are Lowe (1992) and Harris (1993).

These methods are similar in that they both assume an estimate of the current pose is available. This is projected in the image, generating a prediction view, and correspondence is sought between the prediction and the image evidence.

Lowe's method first extracts all the edges from the full image, using the Marr and Hildreth (1980) edge detector. These edges are then connected to form lines. Correspondence is sought between the most reliable lines extracted from the edges and those from the predicted image. Another prediction view is then generated from the new estimation of the pose and further correspondences are sought, between the unmatched lines from the image and the model. The pose update step is a minimisation of a sum of weighted least squares, with prior knowledge (in the form of variance information) used for stabilisation. The multi-step methodology reduces the search space incrementally, using the most likely matches in the image. The disadvantage is that if the first step leads to incorrect matchings, the other step is likely to fail.

Related to this work, Koller et al. (1993) represent the 3D model with line edges, which are defined by length, the coordinates of the middle point and the orientation angle. Pose recovery is done by minimising a function based on the Mahalanobis (1936) distance between all model segments and all data segments, using the Levenberg–Marquardt algorithm (Moré, 1978).

These approaches tend to be unreliable, as they rely heavily on an accurate edge extraction step. Harris' alternative method, called RAPiD, does not begin by extracting all the edges in the image, but rather by generating the prediction view. High contrast model edges are extracted and, from these, several feature points (called *control* points) are selected. Next, for each control point, the algorithm finds the

most prominent image edge perpendicular to the model outline (Figure 2.1), by using a 1D search. The distances between each control point and the closest image edges contribute collectively to a value for the a pose correction, which is recovered using least-squares. The selection of the closest edge can also be augmented with a culling step, which ignores weak edges. Of course, this means that, between a strong and a weak edge close to each other, the algorithm well select the strong one, which might be incorrect. To determine feature visibility, a precomputed table of visible edges is used.

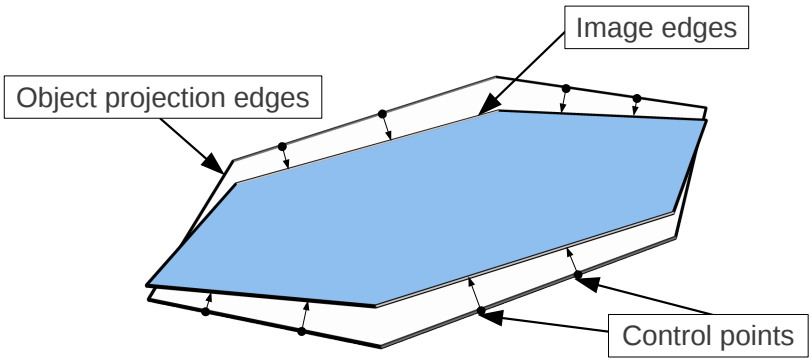


Figure 2.1: RAPiD control points and distances from projected to image edges.

Several improvements have been proposed to RAPiD, on the one hand looking at the edge selection process and, on the other hand, at the pose optimisation algorithm.

For example Armstrong and Zisserman (1995) and Simon and Berger (1998) group control points into primitives (lines, conics, etc.). Armstrong and Zisserman (1995) then use RANSAC (Fischler and Bolles, 1987) to filter the edges detected around each control point. These primitives can also be eliminated from the pose update step. A greedy approach is used: the primitives are eliminated one by one and the pose correction is computed from the remaining ones. If the pose correction leads to a smaller projection error, the primitive is eliminated. Another approach is used by Drummond and Cipolla (2002) where the influence of the control points in the pose optimisation is weighted inverse proportionally to the number of edge strength maxima visible within the search range around the control point. The selection process for the visible edges is also improved by replacing the precomputed indexing table with a 3D rendering stage. Figure 2.2 shows a set of example results from

Drummond and Cipolla (2002).

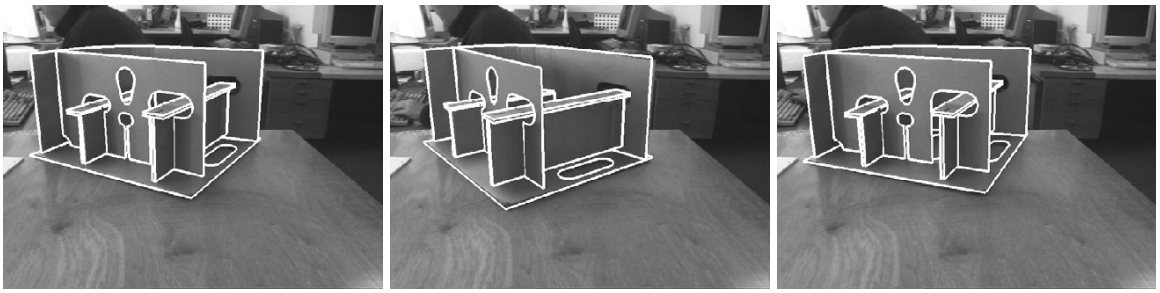


Figure 2.2: Example results from Drummond and Cipolla (2002). Image taken from Drummond and Cipolla (2002), ©2002 IEEE.

Improvements to the pose optimisation algorithm have been brought, among others, by Drummond and Cipolla (2002), Simon and Berger (1998) and Marchand and Bouthemy (2001). Drummond and Cipolla (2002) replace the original least squares approach with a weighted least squares one. Marchand and Bouthemy (2001) split the pose update step in two stages: first the 2D affine transformation between two consecutive frames is estimated; second the standard least squares is replaced by a minimisation of a non linear cost function with respect to the pose parameters. Simon and Berger (1998) use M-estimators for a more robust least squares.

Another improvement, related to the 1D closest point search, was introduced by Comport et al. (2003). Here, rather than doing a blind search for the closest image edge, only the edges with an orientation similar to that of the projection of the 3D model are sought.

A third type of approach was introduced by Kollnig and Nagel (1997) and proposes to remove the edge detection step altogether, and, instead, fit the model projection to the image gradients directly. Theoretically this method avoids the loss of information associated with an edge detection step, but, in fact, it leads to a much smaller basin of convergence. A similar approach was used by Marchand and Bouthemy (2001) as a refinement step.

All the above mentioned methods ignore the inherent multimodality of the image feature-pose mapping i.e. multiple object poses lead to similar image features. To solve this problem one solution is to keep track of multiple poses, rather than a single

best one. One way of doing this is to use a particle filter (Doucet et al., 2001). This approach was introduced to computer vision by Isard and Blake (1998) and used in the context of 3D tracking by, among others, Vacchetti et al. (2004a) and Klein and Murray (2006). One problem with particle filters is that they require a large number of particles to produce accurate results. For deformable object tracking, a solution is presented by MacCormick and Isard (2000), where a technique called *partitioned sampling* is used. Partitioned sampling means dividing the state space into several "partitions" followed by an appropriate weighted resampling operation. The algorithm is applied to hand tracking by modelling the hand as a three joint articulated object and splitting the state space into four sub-spaces, one for scale, orientation and translation of the fist and one for each joint angle. An important advantage of partition sampling is that the number of particles allocated to each partition can be varied. A second advantage becomes obvious: many fewer evaluations of the likelihood functions are necessary. The authors report that using partitioned sampling resulted in twice the speed of normal particle filters.

Regardless of the approach used, all edge based methods are susceptible to motion blur, occlusions and cluttered background, as these lead to the corruption of image edges. Furthermore, as many of these methods rely on line segments for matching, they are also often limited to simplistic 3D models so cannot work with arbitrarily shaped objects.

2.2.2 Point-Based Methods

As with edge based methods, point based methods work by matching a set of features from the image with a set of features on the model. The difference is that, instead of edges, the features they use are interest points. These tend to be more invariant to disturbances (such as changes in scale, viewpoint and illumination), and therefore yield better results. Also, often, they do not require any pose initialisation.

Early examples of point-based methods are Uenohara and Kanade (1991) and Ravela et al. (1995). Rather than keeping a 3D model of the object, these methods use pose-dependent templates images. Patches are selected from both the real and the

template image and are matched using a combination of normalised cross correlation and steerable filters (for invariance to translation and rotation). Such methods do show good results, but are not practical since they require a manual selection of the features.

The rise of effective and automatically discoverable descriptors for point features e.g. Lowe (2004) or Lepetit et al. (2005), has seen real-time methods for point-based model-based tracking become more practical, as shown by, among others Lepetit et al. (2005); Ozuysal et al. (2006); Skrypnyk and Lowe (2004); Vacchetti et al. (2004b) and Wagner et al. (2008). These articles are exponents of a related methodology, called tracking by detection. The template images of Uenohara and Kanade (1991) and Ravela et al. (1995) are replaced by mapping relating feature points to the pose of the object. At run time, features are extracted from the image and matched to entries in the database.



Figure 2.3: Example results from Skrypnyk and Lowe (2004). Image taken from Skrypnyk and Lowe (2004), ©2004 IEEE.

In Skrypnyk and Lowe (2004), for example, the offline phase consists of building a sparse 3D model of the target object, using multiple views and SIFT features (Lowe, 2004). Online, at every frame, SIFT features are first extracted. Next they are matched to the ones from the 3D model using the Best-Bin-First algorithm (Beis and Lowe, 1997), resulting in a collection of 2D to 3D correspondences. The final pose is obtained using a combination of RANSAC and the Levenberg-Marquardt algorithms. The system is too slow for real time performance, running at an average of 4 fps. Figure 2.3 shows example results from Skrypnyk and Lowe (2004). A related method is Lepetit et al. (2005), where a classifier based on randomised trees is used for feature matching. RANSAC is again used for pose recovery. Also, unlike Skrypnyk and Lowe (2004), both accurate 3D geometry of the object and a few images (which are then

used to texture the 3D geometry) are assumed to be known. The requirement for an accurate 3D model is removed in Ozuysal et al. (2006), where the texture and geometry of the object is computed offline using bundle adjustment, with an ellipsoid as an initial shape approximation. Bundle adjustment is also used by Vacchetti et al. (2004b), but this time for tracking, instead of model creation (so here again a 3D model is required). Real time performance on a mobile device is debated by Wagner et al. (2008), where both SIFT and Ferns (Ozuysal et al., 2007) are modified to be usable on a mobile phone.

Assuming a well textured object, point-based methods have been shown to have good accuracy and robustness to occlusions and changes in illumination. Real time performance has been achieved in constrained scenarios. However, issues, such as their inability to cope with motion blur or untextured objects, still exist.

2.2.3 Region Based Methods

Region-based methods such as Rosenhahn et al. (2007); Schmaltz et al. (2007a) and Dambreville et al. (2008b) seek alignment of the projection of the occluding boundaries of the object with a regional segmentation of the image. This is done by maximising the discrimination between a foreground and a background region (with known image statistics), with respect to the 3D pose of the known 3D model.

The curve separating the foreground and the background regions can be represented either *explicitly* or *implicitly*. A conventional explicit representation is to use a collection of equally spaced points, denoted by their $[x, y]$ coordinates. Curves can however breakup, merge, move or disappear during the course of their evolution. Using the explicit representation means that complicated methods have to be developed to model such behaviours. This is not required when using implicit representation, for example level sets (Osher and Sethian, 1988), as they can handle all these topological changes very easily. A level set function Φ (Figure 2.4) is a Lipschitz continuous function, implicitly defining a curve as its zero level. That is to say that, for a 2D closed curve $\mathbf{C}$, we can formally write $\mathbf{C} = \{(x, y) \in \mathbb{R}^2 | \Phi(x, y) = 0\}$. This is called the implicit representation of the curve C. Often, a subset of these implicit functions,

namely *signed distance functions* are used. A signed distance function is a level set function for which $\Phi(\vec{x}) = -d(\vec{x}), \forall \vec{x} \in \Omega_f$ and $\Phi(\vec{x}) = d(\vec{x}), \forall \vec{x} \in \Omega_b$, where:

$$d(\vec{x}) = \min_{\vec{x}_c \in \mathbf{C}} |\vec{x} - \vec{x}_c| \tag{2.1}$$

These also impose an extra condition on the level set function, namely $|\nabla \Phi(x, y)| = 1$.

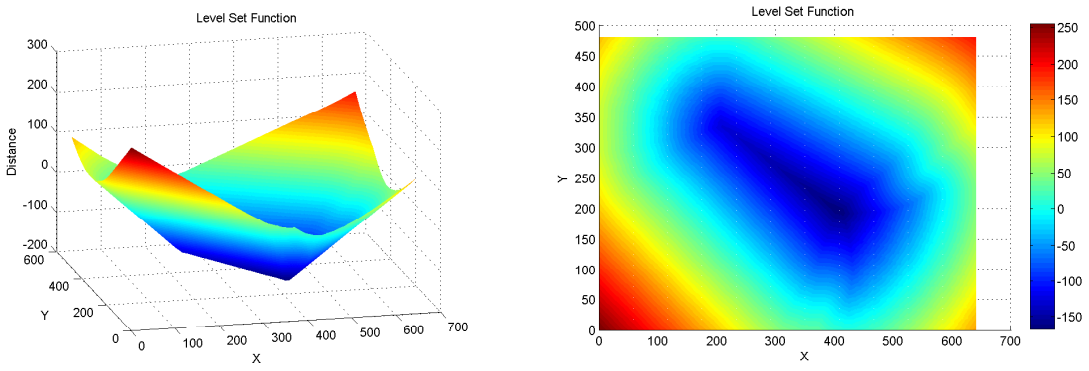


Figure 2.4: Example level set function embedding the contour of the projection of a soft-drink can 3D model. The size of the image containing the contour is 640×480.

Regions were first used by Rosenhahn et al. (2007), where an infinite dimensional active contour is adapted in a single iterative step, in two stages: first the contour, represented by a zero level-set of a 2D embedding function with a shape term (to encourage similitude between the segmentation and the expected silhouette of the object given the current pose), is evolved to find a segmentation, in the expectation that the contour will then match the projection of the occluding contour of the 3D object. Second, each point on the contour is back-projected to a ray (represented in Plücker coordinates (Vallance et al., 1952)), and the pose is sought that best satisfies the tangency constraints that exist between the 3D object and these rays. In some respects it bears similarity to RAPiD system of Harris (1993), with the level set evolution to find the extremal contour replacing Harris' 1D searches for edges. This two-stage iteration places only soft constraints on the evolution of the contour i.e. the minimisation of the energy function is done in an infinite dimensional space (with some encouragement), rather than in the space of possible contours. Furthermore our experience with the two-stage iteration is that the 2D-3D pose matching does

not have a large basin of convergence. The unconstrained contour evolution stage is removed, and the minimisation takes place by evolving the contour (approximately) directly from the 3D pose parameters in Schmaltz et al. (2007a). Here the direction of contour evolution is determined by the relative foreground and background membership likelihoods of each point, while the amount of evolution is apparently a "tunable" parameter. Further extensions to the method are proposed by Gall et al. (2008), where an image synthesis stage is added, and by Brox et al. (2009), where the region statistics are augmented with SIFT features and optical flow. This is at or close to the current state-of-the-art in 3D object tracking and an example result is shown in Figure 2.5.

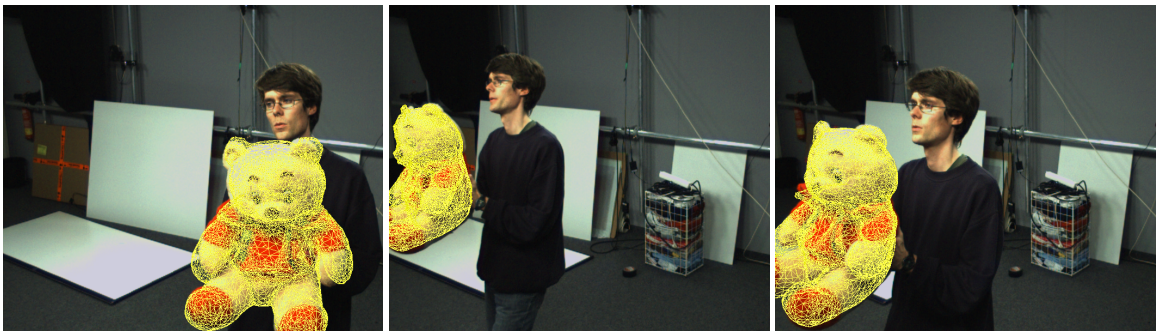


Figure 2.5: Example results from Gall et al. (2008). Images taken from Gall et al. (2008), ©2008 IEEE.

Level sets are not used by Dambreville et al. (2008b). Rather, this work is based on an energy function summing two integrals – one over the foreground, one over the background. Here a direct minimisation over the pose parameters is proposed, by differentiating these integrals with respect to the pose parameters. This is achieved using the Leibniz-Reynolds transport theorem, yielding an integral along the occluding contour and two surface integrals. These surface integrals collapse to zero because the authors choose to represent the statistics of the two regions of the image with the functions used by Vese and Chan (2002).

Regions are less disrupted by occlusions, clutter and motion blur than edge-based methods, making them generally more reliable than both edge and point-based ones. Furthermore, these techniques have been shown to work with arbitrarily shaped ob-

jects. There are two main failure cases, namely they fail when the pose-silhouette mapping is ambiguous (i.e.when multiple poses project to the same silhouette) and when the foreground and background image statistics are too similar or are corrupted.

2.3 Shape Space-Based Segmentation and Tracking

The second class of methods we look at in this thesis, rather than parametrising 2D shape by the 6 DoF pose parameters of a fixed 3D model, attempts to learn a space of shapes. This methodology can have several advantages. Whereas model based approaches are limited in the type of deformations they are able to generate (i.e. only pose dependent deformations), shape space based ones have no such limitations. For example, they can learn all the possible shapes of a class of object, e.g. cars, planes, etc. Alternatively they are able to learn action dependent sets of shapes, such as those caused by a jump or walk.

Shape spaces are inherently very high dimensional. Using implicit shape representations (as detailed in the previous section) and 640×480 images would lead to a 307200 dimensional descriptor. Often however, as it is the case of classes of objects or action dependent deformations, the set of shapes lies on a much lower dimensional manifold. For this reason, often, dimensionality reduction is used to learn the shape space.

We begin with a review of several dimensionality reduction techniques, used throughout the shape space literature, in Section 2.3.1. We review these methods in their generic theoretical form because they are most often used unaltered in the shape space literature. This means that both their advantages and disadvantages will be reflected in the systems that use them. In Section 2.3.2 we detail several shape space-based works. Current work uses either Principal Component Analysis (PCA) or Kernel Principal Component Analysis (KPCA). Our work is based on Gaussian Process Latent Variables Models (GP-LVM), which, to our knowledge has not been

used in segmentation. For this reason in Section 2.3.3 we present a related approach, that of using GP-LVM to learn spaces of joint angles for human pose estimation. While these methods do not lead to an image segmentation, they are useful for pose tracking, and lead to much of the work presented in Chapter 5.

2.3.1 Dimensionality Reduction

In this section we review the main 2 types of dimensionality reduction used in the shape space literature, namely principal component analysis and kernel principal component analysis. We also review Gaussian Process Latent Variable Models, which is the main dimensionality reduction technique used in this thesis.

Principal Component Analysis

Given a data set, Principal Component Analysis (PCA) attempts to find a new basis that maximises the variance of the data along its basis vectors. The greatest variance of the data lies on the first basis vector (which becomes the first *principal component*), the second greatest variance on the second basis vector, etc. This is achieved by computing the eigenvectors of the covariance matrix of the (centered) data set. The eigenvector with the largest eigenvalue is the first principal component, the second largest is the second principal component, etc. Dimensionality reduction is achieved by selecting only a subset of the principal components and projecting the data onto the basis spanned by the reduced set.

PCA is easy to compute and has been successfully used in many (computer vision) tasks, including the one at hand (as detailed in Section 2.3.2). It however lacks a principled probabilistic interpretation, meaning it is difficult to ascertain how likely a low dimensional latent point is to generate a valid data point. PCA is also linear, meaning that, when used on non-linear spaces, a large number of dimensions will be required to capture all the variance. This will inevitably mean that a large amount of non essential variance is captured, leading to a difficult inference.

Kernel Principal Component Analysis

Any non-linear data set $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$ (where each $\mathbf{y}_i$ is a data point of dimensionality d), can be embedded inside a linear, higher D-dimensional space (called a *feature* space), by use of a function $\Phi : R^d \to R^D$. Kernel PCA works by computing standard linear PCA inside this feature space. It does this by defining both a kernel function κ as the inner product inside the feature space $\kappa(\mathbf{y}_i, \mathbf{y}_j) = \Phi(\mathbf{y}_i)^T \Phi(\mathbf{y}_j)$ and a kernel matrix $K_{ij} = \kappa(\mathbf{y}_i, \mathbf{y}_j)$.

As with PCA, KPCA requires the data to first be centered (i.e. it must have zero mean). As shown by Schölkopf et al. (1998) we can define a kernel function which takes into account this requirement, as:

$$\kappa'(\mathbf{y}_i, \mathbf{y}_j) = \kappa(\mathbf{y}_i, \mathbf{y}_j) - \frac{1}{n}\mathbf{1}^T k_x - \frac{1}{n}\mathbf{1}^T k_y + \frac{1}{n^2}\mathbf{1}^T K \mathbf{1} \tag{2.2}$$

where $k_x = [\kappa(\mathbf{y}_i, \mathbf{y}_1), \ldots, \kappa(\mathbf{y}_i, \mathbf{y}_n)]^T$ and $k_y = [\kappa(\mathbf{y}_1, \mathbf{y}_j), \ldots, \kappa(\mathbf{y}_n, \mathbf{y}_j)]^T$.

Analogously to K, we obtain a K' using the centered kernel, $K'_{ij} = \kappa'(\mathbf{y}_i, \mathbf{y}_j)$.

The projection of a point $\mathbf{y}_i$, mapped in the features space as $\Phi(\mathbf{y}_i)$ on the above mentioned kth principal component $\mathbf{V}_k$ becomes:

$$\mathbf{V}_k^T \Phi(\mathbf{y}_i) = \sum_{j=1}^{n} a_{kj} \kappa'(\mathbf{y}_i, \mathbf{y}_j) \tag{2.3}$$

where a_k is the kth eigenvector of K' and a_{kj} is the jth element of a_k.

When κ is defined as the linear kernel, KPCA is linear and equivalent to PCA. Non-linear kernels can also be used, one example being the *Radial Basis Function* (RBF) kernel:

$$\kappa(\mathbf{y}_i, \mathbf{y}_j) = \exp \frac{-(\mathbf{y}_i - \mathbf{y}_j)^2}{2\alpha^2} \tag{2.4}$$

One disadvantage becomes apparent. The parameters of the kernel need to be set manually (α in 2.4), which can easily lead to over or underfitting. Another disadvantage is that the only closed form mapping is from the high dimensional to the low dimensional space. To compute the opposite (known as the *pre-image*), generally, a

non-linear optimisation is required. For example, one way of finding the pre-image (Mika et al., 1999) is to iteratively minimise the following error:

$$E(\hat{\mathbf{y}}) = ||\Phi(\hat{\mathbf{y}}) - P\Phi(\mathbf{y})||^2 \tag{2.5}$$

where $P\Phi(\mathbf{y})$ is the projection of $\Phi(\mathbf{y})$ onto the selected number of eigenvectors (in the feature space).

This method will find a local minimum and is dependent on the starting point. If one assumes an invertible kernel, non-iterative approaches do exist, for example Kwok and Tsang (2003) and Rathi et al. (2006). Kwok and Tsang (2003) use multi-dimensional scaling (MDS) (de Leeuw, 2005), along with a computation of the singular value decomposition of the input data set. This makes it intractable for large data sets with lots of dimensions (as it is our case, where we can easily have matrices of dimension 307200×2000). The second method, Rathi et al. (2006), avoids this calculation, but is based on the assumption that $E(\hat{\mathbf{y}}) \approx 0$ i.e. that for each test point there exists an exact pre-image point. This is obviously not true in many cases.

Gaussian Processes and Gaussian Process Latent Variable Models

Gaussian Processes (GPs) (Rasmussen and Williams, 2005) are distributions over functions. Given some training data, a GP can produce a function approximating the data and point-level covariance estimates.

A GP is defined by a *mean function* $\mu(\cdot)$ and a *covariance function* $\kappa(\cdot, \cdot)$. If a function $f(x)$ is drawn from this GP then for any number of values $X = [x_i]$ the corresponding function values $Y = [f(x_i)]$ are Gaussian distributed, with the distribution:

$$P(Y|X) = \mathcal{N}(Y|\mu(X), \kappa(X, X)) \tag{2.6}$$

The mean function is usually defined to be zero while the covariance function must only be positive semi-definite. The covariance function is equivalent to the KPCA kernel presented in the previous section.

Covariance functions are parametrised with *hyperparameters*, for example α, in

2.4. Training a GP consists of finding values for the hyperparameters so that the training data has a high likelihood. To do this one maximises the following log *marginal likelihood* with respect to the hyperparameters:

$$\log p(y|X,\theta) = -\frac{1}{2}y^T K^{-1} y - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{2.7}$$

where θ are the hyperparameters, n is the dimension of an entry from the data set and K is the covariance matrix of the data set and a function of the hyperparameters θ. Maximisation is achieved by using the derivative of Equation 2.7 with respect to each hyperparameter θ_j.

Gaussian Process Latent Variable Models (GP-LVM) (Lawrence, 2005) is a non-linear dimensionality reduction technique. I.e., just like PCA and KPCA, it allows us to represent a data set $\mathbf{Y} \in R^{n\times d}$, with $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]^T$, where each $\mathbf{y}_i$ is a centered data point of dimensionality d, with a lower dimensional set of latent variables $\mathbf{X} \in R^{n\times q}$ with $\mathbf{Y} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$, where each $\mathbf{x}$ is a latent point of dimensionality q, with $q < d$. GP-LVM finds, by use of a nonlinear optimisation, the values for $\mathbf{X}$ and a Gaussian Process mapping from $\mathbf{X}$ to $\mathbf{Y}$.

GP-LVM can be seen as a probabilistic form of kernel PCA, with both methods using kernels to model non-linearity. The difference is that GP-LVM learns the mapping from the low dimensional space to the high dimensional one whereas KPCA learns the opposite. GP-LVM is also a non-linear generalisation of the probabilistic PCA (Tipping and Bishop, 1999), through the use of GPs.

We first assume a linear relationship between $\mathbf{Y}$ and $\mathbf{X}$:

$$\mathbf{y_i} = \mathbf{W}\mathbf{x}_i + \boldsymbol{\eta}_i \tag{2.8}$$

where $\boldsymbol{\eta}_i$ is Gaussian noise i.e. $\boldsymbol{\eta}_i \sim \mathcal{N}(0, \sigma^2\mathbf{I})$.

The likelihood of any individual observation point $\mathbf{y}_i$ can be written as:

$$P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{W}, \sigma^2) = \mathcal{N}(\mathbf{y}_i|\mathbf{W}\mathbf{x}_i, \sigma^2\mathbf{I}) \tag{2.9}$$

and, assuming independent observation points, for the whole data set, we can write:

$$P(\mathbf{Y}|\mathbf{X}, \mathbf{W}, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{y}_i|\mathbf{W}\mathbf{x}_i, \sigma^2\mathbf{I}) \tag{2.10}$$

We assume a spherical Gaussian distribution on $P(\mathbf{W})$:

$$P(\mathbf{W}) = \prod_{i=1}^{d} \mathcal{N}(\mathbf{w}_i|0, \mathbf{I}) \tag{2.11}$$

with the transposed ith row of the matrix $\mathbf{W}$ being denoted with $\mathbf{w}_i$.

This allows us to integrate out $\mathbf{W}$ and obtain:

$$P(\mathbf{Y}|\mathbf{X}, \sigma^2) = \prod_{i=1}^{d} \mathcal{N}(\mathbf{y}_{:,i}|0, \mathbf{X}\mathbf{X}^\mathbf{T} + \sigma^2\mathbf{I}) \tag{2.12}$$

where $\mathbf{y}_{:,i}$ is the ith column of $\mathbf{Y}$.

This follows in a straightforward manner if we rewrite Equation 2.8 as:

$$\mathbf{y}_{:,i} = \mathbf{X}\mathbf{w}_i + \boldsymbol{\eta}_i \tag{2.13}$$

and compute the covariance as:

$$cov[\mathbf{y}_{:,i}] = E[(\mathbf{X}\mathbf{w}_i + \boldsymbol{\eta}_i)(\mathbf{X}\mathbf{w}_i + \boldsymbol{\eta}_i)^T] = E[\mathbf{X}\mathbf{w}_i\mathbf{w}_i^T\mathbf{X}^T] + E[\boldsymbol{\eta}_i\boldsymbol{\eta}_i^T] = \mathbf{X}\mathbf{X}^T + \sigma^2\mathbf{I} \tag{2.14}$$

The mean is 0 as the data is already assumed to be centered.

Lawrence (2005) shows that the maximum likelihood solution of 2.12, $\mathbf{X}$, is a mapping from the latent space to the principal subspace of the input data set:

$$\mathbf{X} = \mathbf{U}'_q\mathbf{L}\mathbf{V}^T, \mathbf{L} = (\lambda_q - \sigma^2\mathbf{I})^{\frac{1}{2}} \tag{2.15}$$

where $\mathbf{U}'_q$ are the first q eigenvectors of the inner product matrix of the data set and λ_q are their corresponding eigenvalues; and $\mathbf{V}^T$ is an arbitrary orthogonal rotation matrix.

This formulation is called dual probabilistic PCA and is effectively equivalent to probabilistic PCA. Thus, it still has the linearity limitation. The step towards non-linearity is to rewrite equation 2.12 as:

$$P(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{y}_i|0, \mathbf{K}) \tag{2.16}$$

where $\mathbf{K}$ is the covariance matrix of a Gaussian Process, which can be generated by a number of linear and non-linear kernels.

$\mathbf{Y}$ now becomes, instead of a linear function of $\mathbf{X}$ as in Equation 2.12, a non-linear function of $\mathbf{X}$, parametrised by the GP covariance $\mathbf{K}$, itself a function of the hyperparameter of the GP. We can therefore write:

$$\mathbf{y}_i|\mathbf{X} \sim N(\mu_i, \sigma_i^2) \tag{2.17}$$

with

$$\mu_i = \kappa(\mathbf{x}_i, \mathbf{X})K^{-1}\mathbf{Y} \qquad \sigma_i^2 = \kappa(\mathbf{x}_i, \mathbf{x}_i) - \kappa(\mathbf{x}_i, \mathbf{X})K^{-1}\kappa(\mathbf{x}_i, \mathbf{X})^T \tag{2.18}$$

where $\kappa(\cdot, \cdot)$ is the GP kernel function.

GP-LVM is both non-linear and probabilistic. Compared to PCA and KPCA, this means that there exists a principled way of accessing point level covariance. This is particularly useful when searching for latent points that, through backprojection, satisfy conditions in the observation space, as it allows us to restrict the search and results to high probability low dimensional points. Furthermore, the parameters of the mapping are learned instead of manually set. Finally, the non-linear optimisation is used for the learning phase, rather than the generative one, which is closed form (i.e the standard Gaussian Process generative pipeline). On the one hand this allows for much faster, exact, pre-image computations and, on the other hand, it becomes possible to quantify what the latent space has learned.

An extension, known as shared GP-LVM, was introduced simultaneously by Ek et al. (2007) and by Navaratnam et al. (2007). Here, instead of learning a latent space

for a single set of high dimensional variables, joint latent spaces between multiple sets of such variables are learned. This is done by simply concatenating the various data set entries into a single set $\mathbf{Y}$ and using standard GP-LVM on $\mathbf{Y}$. We use this extensively in Chapter 5 and a more detailed introduction can be found there.

2.3.2 Shape Representation

The idea of using high level, statistically modelled, prior knowledge to augment image segmentation was pioneered in works such as Grenander et al. (1991). Here, training contours are first built by manually segmenting several training images. The contours are next split into adjacent segments and probability distributions over the angles between these segments are computed. Such local models cannot however capture global shape information.

Some of the earliest works to capture global shape information comes from Yuille et al. (1992). Here the authors build domain specific models of the human eye using parametrised templates composed from various circles and arcs. Their method is effective, but is however specific to the application for which it was developed.

An early non application specific (and still global) work was introduced by Staib and Duncan (1992). Shapes here were parametrised using elliptic Fourier descriptors (Kuhl and Giardina, 1982) and shape deformations were modelled using independent, multivariate Gaussian distributions (on the Fourier parameters). These could be either learned from sample images or assumed to be uniform (with a given initial approximation of the shape). Example results are shown in Figure 2.6. This particular representation was used in the context of a segmentation, which was cast as a maximum a posteriori (MAP) estimation problem, where the energy function combines the learned shape term (as the prior) and an edge based image matching term. The method is extended by Chakraborty et al. (1994), where regions are used, along with edges. They are not however integrated directly in the MAP estimation, but rather, a region based pre-segmentation is required and the result is fed in a Shaib and Duncan-like optimisation. This method will of course fail when the pre-segmentation step is far from being correct.

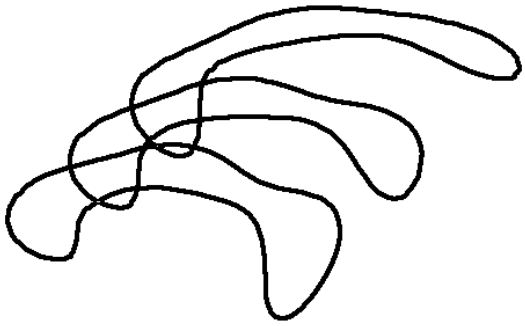


Figure 2.6: Example mean curve (middle) and curves corresponding to plus (above) and minus (below) one standard deviation in the parameters. Image taken from Staib and Duncan (1992), ©1992 IEEE.

A seminal paper in the computer vision literature was Kass et al. (1988). Here shapes are represented by deformable contours called *snakes*, which are collections of corresponding points of interest. In their default formulation, snakes do not have any prior shape information. One of the earliest works to augment snakes with shape knowledge was Cootes et al. (1995) where *Active Shape Models* are introduced. Here, for the first time, PCA is used to learn shape variance. PCA is run on aligned sets of similar shapes (such as those belonging to the same class of objects). To segment an image with the learned space, given an initial approximation of the 2D pose and shape, the authors first estimate displacements for each point in the shape (using a RAPiD (Harris, 1993) like approach) and combine them into global shape and pose changes. The global displacements are then accepted only if they are within three standard deviations of the learned mean shape. Collections of points are also used by Wang and Staib (1998), but, rather than computing eigenvector of the standard covariance matrix, a smoothness covariance matrix is used, which only encodes dependency between neighbouring points.

An alternative formulation of snakes, using B-splines, was proposed, among others, in Curwen and Blake (1993) and Menet et al. (1990). This formulation reduces the number of control points needed to represent the contour and allows for smoothness constraints to be imposed implicitly. Furthermore, B-spline-based snakes have fewer degrees of freedom so they usually tend to be easier to recover. A method for

using shape knowledge in B-spline snakes, by using 2D templates, was proposed in Blake et al. (1993). These templates are invariant to 2D affine transforms, but are unfortunately fixed shape.

The above mentioned methods are constrained by their use of explicit shape representations, which means that they cannot easily model holes or sudden topology changes. Implicit shape representations and especially level set embedding functions (Osher and Sethian, 1988) have been shown to solve these problems. Their use was first explored, as shape priors for segmentation, by Leventon et al. (2000). Here curves are embedded as the zero levels of signed distance functions. To segment the image, the contour is evolved by minimising an energy function which combines an image driven term with a shape term and a pose term, seeking the alignment of the curve with the image gradients, while maximising the probability of the shape.

All the works presented above use edges as the primary segmentation driving image feature. This means they all suffer from the pitfalls of edge based segmentation (lack of robustness to occlusions, motion blur, etc.). Following a pattern similar to the model based literature, whereas early works use edges, later works turn to regions as the main source of image information, these were first used by Tsai et al. (2003). Similar to Leventon et al. (2000), PCA is used on level set functions, but the energy function can be any one of the following: (i) the Chan-Vese one of Vese and Chan (2002), (ii) the Binary Mean Model of Yezzi et al. (1999) and (iii) the Binary Variance Model of Yezzi et al. (1999). The minimisation is done directly in the shape space i.e. by differentiation with respect to the position in the lower dimensional latent space and the 2D pose. Samples from the PCA space are shown in Figure 2.7 and segmentations results in Figure 2.8.

PCA is a linear technique, meaning that all PCA based methods assume the shape space to be linear, which is a rather limiting assumption, as shape spaces are often nonlinear. A series of studies, like Rousson and Paragios (2002) or Cremers et al. (2004), introduced non-linearity in the shape spaces, but at the expense of not using dimensionality reduction. The probability distributions on the space of embedding functions is modelled with Gaussians by Rousson and Paragios (2002) and with a

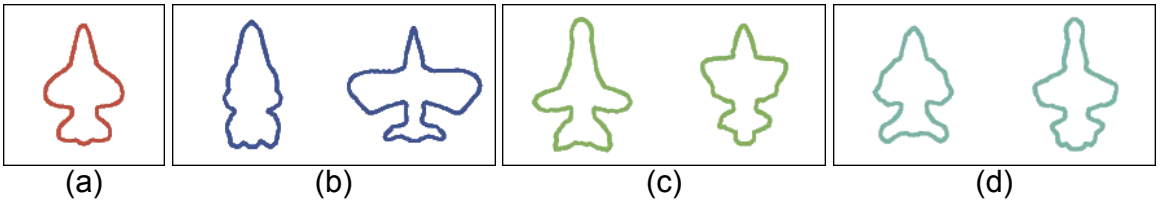


Figure 2.7: Example shape variability for a fighter jet. (a) – mean image; (b) $+1\sigma$ (left) and -1σ (right) variability on the first mode of variation; (c) $+1\sigma$ (left) and -1σ (right) variability on the second mode of variation; (d) $+1\sigma$ (left) and -1σ (right) variability on the third mode of variation. Images taken from Tsai et al. (2003), ©2003 IEEE.

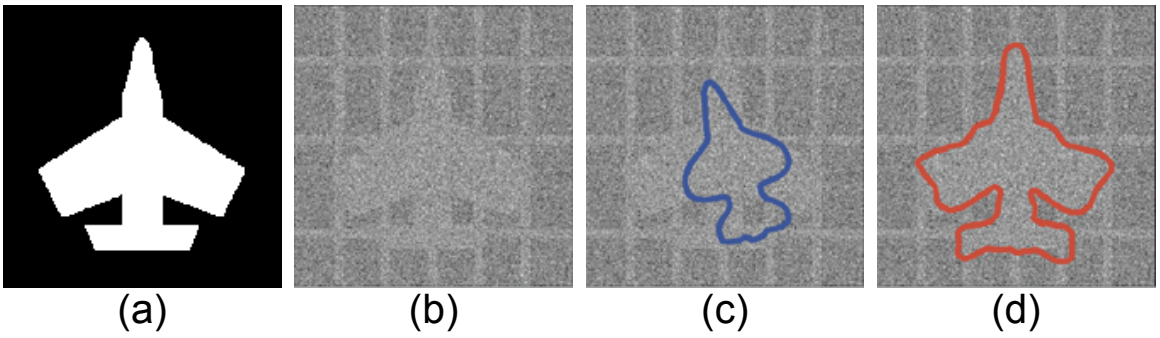


Figure 2.8: Example segmentation for a fighter jet. (a) – original image; (b) – original image with line occlusions and Gaussian noise; (c) – initialisation contour; (d) final result. Images taken from Tsai et al. (2003), ©2003 IEEE.

kernel density estimator by Cremers et al. (2004). Similar to the linearity assumption of PCA, the Gaussianity assumption of Rousson and Paragios (2002) is also rather limiting. While no such limitation affects Cremers et al. (2004) , it does not use dimensionality reduction, making inference difficult. Non-linearity is assumed at the level of shape dynamics, by Cremers and Funka-lea (2006), but PCA is again used.

Nonlinear (but not probabilistic) dimensionality reduction is used by Dambreville et al. (2008a) (which is, to our knowledge, close to or at the current state-of-the-art), in the form of Kernel PCA. This has been shown to greatly improve the learning capabilities of the shape space, as shown in Figure 2.9. However, since KPCA learns a mapping from the higher dimensional space to the lower dimensional one, Dambreville et al. (2008a) optimises an energy function in the space of embedding functions, aiming to minimise the distance between the projection of those embedding functions and the known lower dimensional points. This means that the evolution of the con-

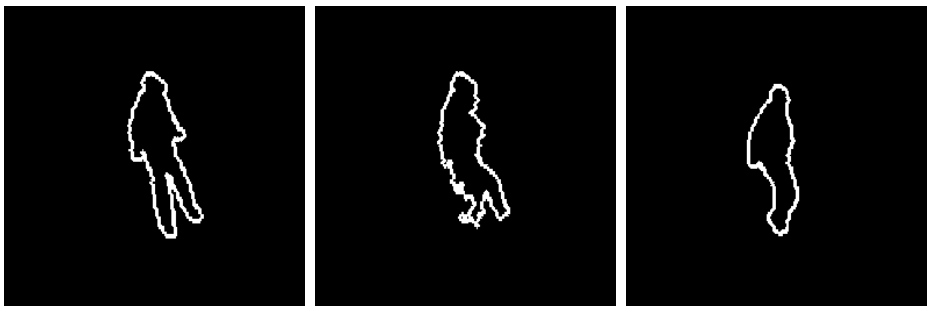


Figure 2.9: Improvements brought by the use of KPCA over PCA. (a) – initial image; (b) – results when using PCA; (c) – result using KPCA. The KPCA results is much more realistic than the PCA result. Image taken from Dambreville et al. (2008a), ©2008 IEEE.

tour is not constrained to high probability shapes. The generative process itself is a nonlinear optimisation, making it slow and susceptible to local minima. Finally, Dambreville et al. (2008a) need the parameter of the embedding kernel to be set manually, which can lead to overfitting: a small value for this parameter produces high accuracy but at the cost of a reduced basin of convergence, while a large value leads to the opposite.

The above mentioned methods are limited to 2D shapes. Sandhu et al. (2009) and Sandhu et al. (2011) are the only ones, to our knowledge, to extend the latent shape space technique to 3D shapes (represented using 3D level set functions), first using PCA in Sandhu et al. (2009) and then using KPCA in Sandhu et al. (2011). The 3D shape recovery is combined with the 3D pose optimisation from Dambreville et al. (2008b) i.e. using the Leibniz-Reynolds transport theorem and the simple image statistics of Vese and Chan (2002). Apart from suffering from the same pitfalls as most PCA and KPCA methods (e.g. limited representational ability or proneness to overfitting and small basins of convergence), these methods are also very limited in the number of training data points they can use. For example, the data set used by Sandhu et al. (2011) only has 12 shapes. This is a direct result of their very high memory requirements e.g. holding in memory a small dataset of just 100 3D shapes (discretised as voxel cubes of dimensionality $256 \times 256 \times 256$) requires 6.4GB of RAM memory.

2.3.3 GP-LVM for Articulated Pose Tracking

As with shape spaces, the human pose space has a very high dimensionality (tens of dimensions) but the positions of the various parts are highly correlated. While (to the best of our knowledge) no prior GP-LVM based segmentation approaches exist, GP-LVM has been used extensively for human pose tracking, with very good results. We review some of this work in this section, as it is highly relevant to our work from Chapter 5. A full survey of these and other methods for human pose estimation is beyond the scope of this work. The interested reader is referred to Moeslund et al. (2006) for a more detailed review.

Pose estimation can usually be categorised, among others, as being generative or discriminative and activity specific or activity independent. Generative approaches learn (low dimensional) representations from which the observed data can be generated while discriminative approaches learn non-parametric models for regression. Activity specific approaches assume the activity which leads to the pose (i.e. playing golf, skying, walking) is known, while activity independent approaches do not make this assumption. GP-LVM methods are usually generative and activity specific.

The first paper to use GP-LVM for pose tracking was Urtasun et al. (2005). Here interest points (position of chest, knees, head, etc.) were tracked during a specific activity (like playing golf) and GP-LVM was used to learn the low-dimensional embedding of high-dimensional human pose data. To perform tracking an initial guess of the pose and its latent position are assumed to be known. Then, at each frame, standard optimisation techniques were used to minimise a log posterior: the probability of the pose, given the image sequence and the learned latent mapping. The method has been improved upon by Urtasun et al. (2006) where the use of dynamics was introduced. The same method is adapted by Shen et al. (2009) to deal with incorrect and corrupted feature data, i.e. human silhouettes affected by shadows, motion blur, etc. Here, the low-dimensional manifold is split into several sub-manifolds, each corresponding to a component of the silhouette and a probabilistic method is developed to fuse the information from the uncorrupted components and

ignore the corrupted ones.

All of these methods are unimodal. One way of modelling multimodality is to learn the probability distribution of the pose, given a silhouette. This is done in Navaratnam et al. (2007) and Ek et al. (2007), where the standard GP-LVM technique is used to learn joint pose-feature descriptor spaces. Ek et al. (2007) call this method *shared GP-LVM*. Navaratnam et al. (2007) also show that these spaces can be trained using unlabelled data i.e. samples of *p(pose)* and *p(image feature)*. Although these methods are able to deal with multimodality, they do not explicitly model ambiguous feature representation. Ek et al. (2008) next proposed an extended formulation of canonical correlation analysis (CCA), called *non-consolidating canonical analysis* (NCCA), to explicitly model multimodality. Like normal CCA, they construct a shared latent space describing the correlation between pose and image feature, but, unlike CCA, they also build two other latent spaces, one of which contains only the information that is unique to pose which cannot be described by the silhouette and the other contains the information that is unique to silhouette but cannot be described by the pose. The mapping between latent and data space becomes a function of both the shared space and the unique spaces. The method of Ek et al. (2008) has one further advantage: initialisation has a closed form which allows the globally optimum solution to be obtained, avoiding the pitfalls of local optimisation.

2.4 Conclusions

There has been much work on shape knowledge in segmentation and tracking, but there are still major issues that need to be tackled.

A first type of methods, i.e. the ones that capture 2D object shape as projection of known, rigid, 3D shape, do not generally work well in real world environments (cluttered, with occlusions and motion blur). There is a wide spread use of edges, which leads to an inability to deal with complex backgrounds. Point features are more robust, but often lead to very slow algorithms. Very few algorithms track regions, which would eliminate a lot of the background related problems. The few

that do, are still either multiple stage (extract segmentation, match to projection, adjust segmentation) or can only use simplistic image statistics. Finally, all region based method also tend to be very slow.

A second type of methods, i.e. the ones that learn the space of 2D object shapes from examples, are often region based, but are generally limited by their use of linear dimensionality reduction. This, on the one hand, means that they require (relatively) large numbers of dimensions to capture the required variance to generate accurate shapes. On the other hand, it means that they are unlikely to be able to generate anything other than shape (e.g. pose), as joint spaces tend to be highly nonlinear. Some papers do use nonlinear dimensionality reduction, in the form of Kernel PCA. These however cannot gauge how likely a generated shape is to be valid, are slow and susceptible to local minima and to overfitting. Nonlinear and probabilistic dimensionality reduction can be done using Gaussian Process Latent Variable Models, but, to our knowledge, this method has not been used to learn shape spaces. It has however shown good results in human pose recovery.

In the next chapter we introduce our novel 3D region-based tracker which can track 3D rigid objects in real-time even in cluttered backgrounds and when being subjected to motion blur and occlusions. We expand this tracker in Chapter 4 and show two applications, driver assistance and hand tracking. Finally, in Chapter 5 we use GP-LVM latent spaces of 2D and 3D shapes and joint shape – parameter spaces, and show that this methodology produces results at or close to the current state of the art in multiple fields, such as gaze tracking and articulated pose recovery.

Chapter 3

PWP3D: Real-time Segmentation and Tracking of 3D Objects

This chapter presents our model based 3D tracker. It works by minimising a level set and region based energy function, measuring discrimination between image foreground and background, with respect to the 6 degree of freedom pose of a known, fixed 3D model. As image statistics, we use pixel wise posteriors upon which we enforce temporal consistency. We further show how our algorithm extends to tracking multiple objects from multiple views and explain why and how invisible points on the model should be used. Finally, we propose a real time, GPU based, implementation. This work was published in Prisacariu and Reid (2009b) and Prisacariu and Reid (2012b).

3.1 Introduction

A first class of methods that uses shape information works on the assumption that, given an accurate 3D model of an object, its segmentation from any given image is fully defined by its pose. In this chapter we develop such a method, by seeking the six degree of freedom rigid transformation that maximises an energy function based on the posterior foreground and background membership probabilities of each pixel in the image, thus computing the pose and segmentation of the object simultaneously,

using a single, unified energy function. We describe an implementation that achieves real-time performance.

Our method is level set and region-based. Inspired by Bibby and Reid (2008), we aim to maximise the posterior per-pixel probability of foreground and background membership, as a function of pose. Bibby and Reid (2008) showed that this yields a better behaved energy function (in the 2D case), when compared to standard level set formulations such as Cremers et al. (2007). As in Rosenhahn et al. (2007) and Schmaltz et al. (2007a), we assume a known 3D model and one or more calibrated cameras, and seek the six degrees of freedom rigid transformation that maximises the pixel-wise posterior energy function. Unlike Rosenhahn et al. (2007) and Schmaltz et al. (2007a), the minimisation is fully probabilistic and variational, similar in spirit to Dambreville et al. (2008b). However, rather than formulate our energy over the sum of separate integrals for foreground and background (as done by Dambreville et al. (2008b)), we use the Heaviside step function, evaluated on the 2D embedding function of the contour, to delineate foreground and background in the integral. This has two significant advantages: (i) the maths is simpler, and permits greater flexibility in the choice of optimisation method while still allowing for real-time speeds; and (ii) more importantly, it is trivial to replace the Heaviside step function with a blurred version, and in consequence we can very naturally allow for a degree of uncertainty in the location of the contour.

The remainder of the chapter is structured as follows: we begin, in Section 3.2, by briefly describing the geometry and setting up our notational conventions. In Section 3.3 we describe the mathematical foundations, while in Section 3.4 we derive our algorithm and discuss its implementation. In Section 3.5 we consider various qualitative and quantitative evaluations of our method, on both real and synthetic data. We conclude in Section 3.6.

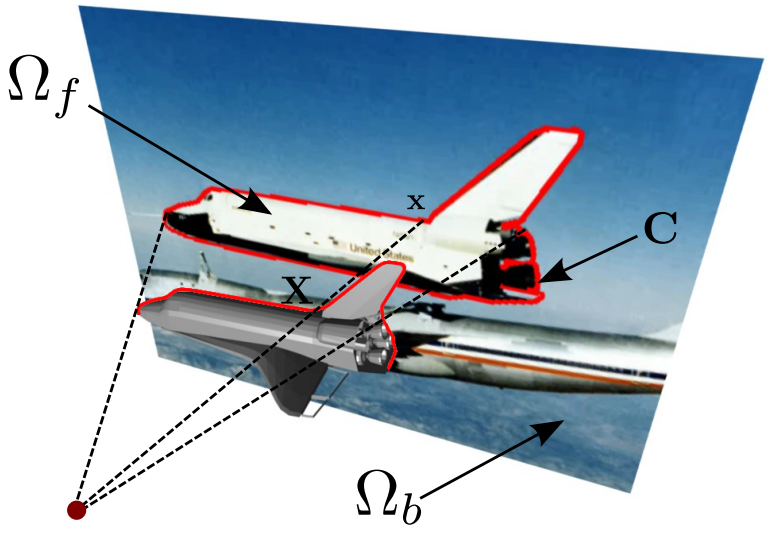


Figure 3.1: Representation of the object showing: the contour of the projection $\mathbf{C}$ and the corresponding contour around the visible part of the 3D model, the foreground region Ω_f and the background region Ω_b, a 2D point $\mathbf{x}$ on the contour and its corresponding 3D point in the object coordinate frame $\mathbf{X}$.

3.2 Notation

Let the image be denoted by $\mathbf{I}$ and the image domain by $\Omega \subset \mathbb{R}^2$. An image pixel $\mathbf{x} = [x, y]$ has a corresponding image value $\mathbf{I}(\mathbf{x}) = \mathbf{y}$ (in our experiments an RGB value), a 3D point $\mathbf{X} = [X, Y, Z]^T = \mathbf{R}\mathbf{X_0} + \mathbf{T} \in \mathbb{R}^3$ in the camera coordinate frame and a point $\mathbf{X_0} = [X_0, Y_0, Z_0]^T \in \mathbb{R}^3$ in the object coordinate frame. $\mathbf{R}$ and $\mathbf{T}$ are the rotation matrix and translation vector respectively (representing the unknown pose) and are parametrised by 7 parameters (4 for rotation and 3 for translation), denoted by λ_i. We use quaternions to represent rotation, so we use 4 parameters to capture the 3 degrees of freedom of the rotation.

We assume the camera calibration parameters (for each camera) to be known. Let (f_u, f_v) be the focal distance expressed in horizontal and vertical pixels and (u_o, v_o) the principal point of the camera. In the case of multiple cameras, the extrinsics were obtained using the Matlab Calibration Toolbox (Bouguet, 2008).

The contour around the visible part of the object in 3D projects to the contour $\mathbf{C}$ in the image (marked with red in Figure 3.1). We embed $\mathbf{C}$ as the zero level-set

of the function $\Phi(\mathbf{x})$ (Osher and Sethian, 1988). The contour $\mathbf{C}$ also segments the image into two disjoint regions: foreground denoted by Ω_f and background denoted by Ω_b. Each region has its own statistical appearance model, $P(\mathbf{y}|M_f)$ for foreground and $P(\mathbf{y}|M_b)$ for background, here implemented using 32 bin histograms.

Finally, we denote the smoothed Heaviside step function by $H_e(x)$ and the smoothed Dirac delta function by $\delta_e(x)$. $H_e(x)$ is obtained using the logistic function and $\delta_e(x)$ is the derivative of the logistic function.

3.3 Segmentation and Pose Tracking

It is common in region-based segmentation for a closed curve to be evolved such that the discrepancy between the statistics of the foreground region and those of the background region is maximised:

$$E = \int_{\Omega_f} r_f(I(\mathbf{x}), \mathbf{C}) d\Omega + \int_{\Omega_b} r_b(I(\mathbf{x}), \mathbf{C}) d\Omega \tag{3.1}$$

where r_f and r_b are two monotonically decreasing functions, measuring the matching quality of the image pixels with respect to the foreground and background models.

In level-set region-based segmentation (Cremers et al., 2007) the energy function is rewritten as:

$$E(\Phi) = \int_{\Omega} H_e(\Phi) r_f(\mathbf{x}) + \big(1 - H_e(\Phi)\big) r_b(\mathbf{x}) d\Omega \tag{3.2}$$

As stated before, the level-set embedding function $\Phi(\mathbf{x})$ is taken to be a signed distance function (or an approximation thereof).

Typically, r_f and r_b are given by the likelihood of a pixel property (such as colour) under a given model, i.e. $r(\mathbf{x}) = P(\mathbf{y}|M)$, with M being either M_f or M_b. In contrast, Bibby and Reid (2008) proposed a generative model of image formation that leads to a subtly different (and more effective) energy function, which can be interpreted in terms of the posterior (as opposed to likelihood) of each pixel's foreground or background membership. Replacing integration with summation, the energy given

by the negative log matching score of the shape of the contour (encoded by the embedding function Φ), given the image data, is:

$$P(\Phi|\mathbf{I}) = \prod_{\mathbf{x}\in\Omega} \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b \Big) \Rightarrow \tag{3.3}$$

$$E(\Phi) = -\log(P(\Phi|\mathbf{I})) = -\sum_{\mathbf{x}\in\Omega} \log \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b \Big) \tag{3.4}$$

where P_f and P_b are defined as:

$$P_f = \frac{P(\mathbf{y}|M_f)}{\eta_f P(\mathbf{y}|M_f) + \eta_b P(\mathbf{y}|M_b)} \qquad P_b = \frac{P(\mathbf{y}|M_b)}{\eta_f P(\mathbf{y}|M_f) + \eta_b P(\mathbf{y}|M_b)} \tag{3.5}$$

with η_f and η_b being the areas of the foreground and background regions respectively:

$$\eta_f = \sum_{\mathbf{x}\in\Omega} H_e\big(\Phi(\mathbf{x})\big) \qquad \eta_b = \sum_{\mathbf{x}\in\Omega} \Big(1 - H_e\big(\Phi(\mathbf{x})\big)\Big) \tag{3.6}$$

In the standard level set formulation, the contour would be evolved in an unconstrained space, so we would compute the derivative of $E(\Phi)$ with respect to time. Instead, we differentiate with respect to the pose parameters λ_i, aiming to evolve the contour in a space parametrised by them:

$$\frac{\partial E}{\partial \lambda_i} = -\sum_{\mathbf{x}\in\Omega} \frac{P_f - P_b}{H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b} \frac{\partial H_e(\Phi)}{\partial \lambda_i} \tag{3.7}$$

$$\frac{\partial H_e\big(\Phi(x,y)\big)}{\partial \lambda_i} = \frac{\partial H_e}{\partial \Phi} \left(\frac{\partial \Phi}{\partial x}\frac{\partial x}{\partial \lambda_i} + \frac{\partial \Phi}{\partial y}\frac{\partial y}{\partial \lambda_i} \right) = \delta_e(\Phi) \begin{bmatrix} \dfrac{\partial \Phi}{\partial x} & \dfrac{\partial \Phi}{\partial y} \end{bmatrix} \begin{bmatrix} \dfrac{\partial x}{\partial \lambda_i} \\ \dfrac{\partial y}{\partial \lambda_i} \end{bmatrix} \tag{3.8}$$

Note that P_f and P_b depend on the ratio between the η_f and η_b areas, so they potentially change with the shape Φ and the pose λ, in particular with the pose parameter λ governing scale. However, as an (infinitesimally) small change in pose leads to an (infinitesimally) small change in the ratio between the foreground and background areas, the derivatives of P_f and P_b are numerically very small. For ex-

ample, the total value of any $\frac{\partial P_f}{\partial \lambda_i}$ is smaller than that of a single per point derivative of the rest of the energy function (i.e. Equation 3.7). We therefore ignore them in the derivation.

At each iteration of the algorithm we re-compute Φ as the signed-distance function from the projected contour. The differentials $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$ follow trivially, using centred finite differences.

Every 2D point $\mathbf{x}$ on the contour of the (true) projection of the 3D model has at least one corresponding 3D point $\mathbf{X}$. For each of these points we can write:

$$\begin{bmatrix} x \\ y \end{bmatrix}^T = \begin{bmatrix} f_u \frac{X}{Z} + u_0 \\ f_v \frac{Y}{Z} + v_0 \end{bmatrix}^T \tag{3.9}$$

Therefore:

$$\begin{aligned} \frac{\partial x}{\partial \lambda_i} &= f_u \frac{\partial}{\partial \lambda_i} \frac{X}{Z} = f_u \frac{1}{Z^2} \left(Z \frac{\partial X}{\partial \lambda_i} - X \frac{\partial Z}{\partial \lambda_i} \right) \\ \frac{\partial y}{\partial \lambda_i} &= f_v \frac{\partial}{\partial \lambda_i} \frac{Y}{Z} = f_v \frac{1}{Z^2} \left(Z \frac{\partial Y}{\partial \lambda_i} - Y \frac{\partial Z}{\partial \lambda_i} \right) \end{aligned} \tag{3.10}$$

λ_i	$\frac{\partial X}{\partial \lambda_i}$	$\frac{\partial Y}{\partial \lambda_i}$	$\frac{\partial Z}{\partial \lambda_i}$
t_x	1	0	0
t_y	0	1	0
t_z	0	0	1
q_x	$2q_yY_0 + 2q_zZ_0$	$2q_yX_0 - 4q_xY_0 - 2q_wZ_0$	$2q_zX_0 + 2q_wY_0 - 4q_xZ_0$
q_y	$2q_xY_0 - 4q_yX_0 + 2q_wZ_0$	$2q_xX_0 + 2q_zZ_0$	$2q_zY_0 - 2q_wX_0 - 4q_yZ_0$
q_z	$2q_xZ_0 - 2q_wY_0 - 4q_zX_0$	$2q_wX_0 - 4q_xY_0 + 2q_yZ_0$	$2q_xX_0 + 2q_yY_0$
q_w	$2q_yZ_0 - 2q_zY_0$	$2q_zX_0 - 2q_xZ_0$	$2q_xY_0 - 2q_yX_0$

Table 3.1: Partial differentials of the 3D point in the camera frame $\mathbf{X}$ with respect to the pose parameters.

We choose to parametrise the rotation with quaternions. While this has the disadvantage of being an overparametrisation, and with requirement to normalise the parameters periodically to 1 (we do this after each frame), it resulted in better performance over using the axis-angle representation.

Every 3D point $\mathbf{X}$ in camera coordinates has a corresponding 3D point $\mathbf{X_0}$ in

object coordinates:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \\ 0 \end{bmatrix} \tag{3.11}$$

with

$$\mathbf{R} = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w & 0 \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w & 0 \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

where q_x, q_y, q_z, q_w are the quaternion rotation parameters and t_x, t_y, t_z are translation parameters.

Finally, the derivatives of the 3D points in camera space with respect to the pose parameters are shown in Table 3.1.

3.3.1 Optimisation Algorithm

There are several optimisation methods at our disposal. Here we detail our results with gradient descent and conjugate gradients.

The obvious first choice is gradient descent. This method uses only the first derivative, yielding fast iterations. Unfortunately a very small step size must be used to avoid jumping over minima. We use a step size of approximately 0.5 degrees for rotation and 5 pixels in either direction (including z scaled with focal length), for translation. These step sizes were chosen experimentally.

The nonlinear conjugate gradients method uses only the first derivative (like gradient descent) but instead of going downhill in the direction of the local gradient, it uses the conjugate direction. To obtain the conjugate of the current direction, we follow a Gram-Schmidt process and define a β function and a direction d_k at step k as $d_k = -g_k + d_{k-1}\beta_k$, where g_k is the value of the derivative of the function to be minimised and d_{k-1} is the direction used at the previous step. The two most popular choices (Hager and Zhang, 2005) for this function are the Polack-Ribiere $\beta_k^{PR} = \frac{\langle g_k, g_k - g_{k-1} \rangle}{\|g_{k-1}\|^2}$ and the Fletcher-Reeves $\beta_k^{FR} = \frac{\|g_k\|^2}{\|g_{k-1}\|^2}$ formulae, where $\langle \cdot, \cdot \rangle$ is the scalar product, $\|\cdot\|^2$ is the L_2 norm and g_k and g_{k-1} are the derivatives of the function

to be minimised at the current and previous iterations. Throughout our testing we noted that the Polack-Ribiere formula leads to faster convergence in the case of rotation, but the Fletcher-Reeves function often leads to faster convergence in the case of translation. We did not observe the Fletcher-Reeves jamming phenomenon (Hager and Zhang, 2005), but we did have cases where the Polack-Ribiere formula did not lead to a correct minimum. In this work we use the hybrid β function proposed in Gilbert and Nocedal (1992) which, throughout our testing, provided the best results of both β_k^{FR} and β_k^{PR} while still yielding convergence: $\beta_k^{GN} = \max\{-\beta_k^{FR}, \min\{\beta_k^{PR}, \beta_k^{FR}\}\}$.

We do not use any preconditioning. The standard preconditioner used in nonlinear conjugate gradients is the Hessian matrix, or an approximation obtained by a Broyden-Fletcher-Goldfarb-Shanno (BFGS) iteration (Press, 2007). Since a preconditioner needs to be positive definite, we need to test if our Hessian matrix is positive definite at every iteration. This, along with the computation of the Hessian matrix, increases iteration time considerably, without leading to a significant reduction in the number of iterations required for convergence.

Another problem with nonlinear conjugate gradients is that if the function is not close to quadratic, iterations tend to lose conjugacy and require a reset in the steepest gradient direction. This is usually done after a preset number of iterations. In this work we evaluate (at every iteration) the energy function after a gradient descent iteration and after a conjugate gradient descent iteration and choose the one which leads to the smallest value for the energy function. This guarantees fast convergence but increases the computation time for each iteration, as the energy function needs to be evaluated twice at each iteration.

The number of iterations needed for convergence depends on the algorithm used, on the complexity of the 3D model, on the step size and, of course, on the distance between the current pose estimate and the final pose estimate. For a simple model (with few triangles), like the one shown in Figure 3.2, when using gradient descent, the algorithm typically converges in up to 8 iterations per frame (in the case of a fixed step) or 2-3 when a line search is used. More complex models require more iterations. For example, for a 3D model of a hand, like the one in Figure 3.4, the algorithm

typically converges within 20 simple gradient descent iterations. Since our algorithm is region-based, the convergence speed is dictated by the size of the regions. If the pose is defined by a small feature on the silhouette, e.g. the tip of a finger from a hand, convergence will be slower, because the area of overlap will be smaller.

The speed of convergence also depends on how close the energy function is to a quadratic function. When recovering translation (i.e. when the energy function is close to being quadratic) we observed that conjugate gradients achieves convergence about 3 times faster than gradient descent. In the rotation case (when the energy function is far from being quadratic) the conjugate gradients needs more resets and actual convergence is slower (in time) than gradient descent, even though the number of iterations is still smaller. In Table 3.2 we depict our timing results for the translation and rotation cases on a sample case where both rotation and translation were very far from the correct values (50 degrees on the x and y axis and 70 degrees on the z axis for rotation and about 5% overlap for translation). Note that this is an extreme case and usually, for tracking, the starting point would typically be an order of magnitude or more closer to the true value, with corresponding less computational effort (see Section 3.5). The setup for this experiment is depicted in Figure 3.2.

	Translation				Rotation			
	GD	GD-LS	CG	CG-LS	GD	GD-LS	CG	CG-LS
Iteration no.	120	40	39	36	132	60	57	54
Iter. time (ms)	4.45	12.2	13.5	14.1	4.65	12.6	13.8	14.2
Total time (ms)	534	489	528	508	613	760	791	769

Table 3.2: Number of iterations and timing details required for convergence when using gradient descent with (*GD*) and without (*GD-LS*) a line search step and conjugate gradients with (*CG*) and without (*CG-LS*) a line search step. The time per iteration for translation is smaller than that for rotation because computing the derivatives with respect to rotation requires considerably more mathematical operations (about 10 times more). The setup for this experiment is depicted in Figure 3.2.

To summarise, while conjugate gradients does often lead to a smaller number of iterations (as high as 3 times less iterations than gradient descent), it also takes 2-3 times the processing time of gradient descent, per iteration. In our experiments we

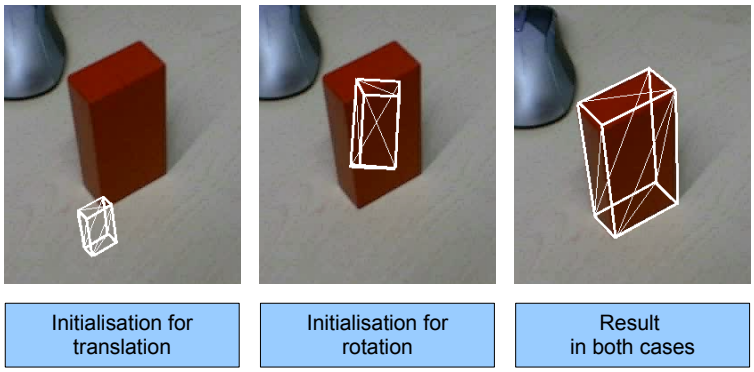


Figure 3.2: Setup (image, model used and initialisations) for the convergence experiment.

use gradient descent unless otherwise stated.

To check for convergence we consider the magnitude of the derivatives of the energy function with respect to the pose parameters. When it is smaller than a preset threshold (15 in our tests) we stop iterating.

3.3.2 Multiple Objects

To track multiple objects we minimise a separate energy function (Equation 3.4) for each object. Each object has its own level set embedding function Φ and foreground and background membership probability (therefore also its own normalising factors η_f and η_b and foreground and background histograms, $P(\mathbf{y}|M_f)$ and $P(\mathbf{y}|M_b)$). For an object j, the object-level energy function becomes:

$$E^j(\Phi^j) = -\sum_{\mathbf{x}\in\Omega} \log\left(H_e(\Phi^j)P_f^j + \big(1 - H_e(\Phi^j)\big)P_b^j\right) \tag{3.13}$$

When tracking multiple objects quite often they occlude each other, or at least parts of each other. If pixels belonging to an object are considered in the energy function of another object, an incorrect pose will be obtained for the second object. Since we extract the full 6 DoF pose of each object, reasoning about the relative depths and occlusions in the multi-object case is trivial. We use the current estimate of the poses to render all tracked objects and select only the visible pixels from each 2D projection, in a manner similar to Schmaltz et al. (2007b).

3.3.3 Multiple Views

Our system can use multiple views of the same 3D object(s) to achieve better tracking. All the pixels from all the views, contribute towards the minimisation of a *single* energy function (for each object), yielding a *single* set of pose parameters. There are however different foreground and background models and level set functions. For V views and a view v, the energy function becomes:

$$E(\Phi^1, \ldots, \Phi^V) = -\sum_{v=1}^{V} \sum_{\mathbf{x}\in\Omega} \log\left(H_e(\Phi^v)P_f^v + \big(1 - H_e(\Phi^v)\big)P_b^v\right) \tag{3.14}$$

When tracking multiple objects from multiple views, the object-level energy function becomes:

$$E^j(\Phi^{j,1}, \ldots, \Phi^{j,V}) = -\sum_{v=1}^{V} \sum_{\mathbf{x}\in\Omega} \log\left(H_e(\Phi^{j,v})P_f^{j,v} + \big(1 - H_e(\Phi^{j,v})\big)P_b^{j,v}\right) \tag{3.15}$$

where j is the object and v is the view.

We obtained the intrinsic camera parameters for each camera and the extrinsic parameters between the cameras using the Matlab Calibration Toolbox (Bouguet, 2008).

3.3.4 Invisible Points

When a 3D model is rotated or translated, surfaces may appear or disappear, as the pose transitions between so-called "generic" views (Binford, 1981; Freeman, 1993). The singular poses between generic views are characterised by ambiguous back-projections: a back-projected image contour point may have multiple tangencies with the object. In these cases the rate of change of the image contour with respect to the pose parameters is not only dictated by the closest points on the 3D model (from the camera) but also by the (infinitesimally invisible) furthest points (see Figure 3.3). In these transition cases we include terms from *both* the closest and the most distant points to give greater robustness when moving between different generic views of the

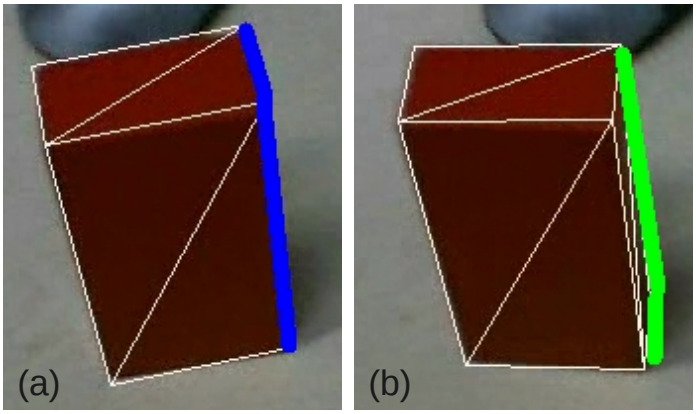


Figure 3.3: Importance of considering the most distant points. As the camera moves from (a) to (b) the evolution of the silhouette is not dictated by the blue points (closest points) but rather by the green points, which were the most distant points in (a).

object. To our knowledge this aspect has not been considered in any of the previous literature on region-based 3D tracking.

Equation 3.10 becomes:

$$\begin{aligned}\frac{\partial x}{\partial \lambda_i} &= f_u \frac{1}{Z_c^2}\left(Z_c \frac{\partial X}{\partial \lambda_i} - X \frac{\partial Z_c}{\partial \lambda_i}\right) + f_u \frac{1}{Z_d^2}\left(Z_d \frac{\partial X}{\partial \lambda_i} - X \frac{\partial Z_d}{\partial \lambda_i}\right) \\ \frac{\partial y}{\partial \lambda_i} &= f_v \frac{1}{Z_c^2}\left(Z_c \frac{\partial Y}{\partial \lambda_i} - Y \frac{\partial Z_c}{\partial \lambda_i}\right) + f_v \frac{1}{Z_d^2}\left(Z_d \frac{\partial Y}{\partial \lambda_i} - Y \frac{\partial Z_d}{\partial \lambda_i}\right)\end{aligned} \tag{3.16}$$

where Z_c and Z_d are the depth values for the closest point and most distant points in the model that project to $[x, y]$.

While we do acknowledge that only either the closest or the most distant point (not both of them) may influence the contour at any given time, checking all possible hypotheses would be intractable, because for each pixel in the 3D rendering both hypotheses would need to be checked, leading to a total of 2^N hypotheses, where N is the total number of pixels inside the 3D rendering. Our solution provides a good approximation, as proven empirically in Section 3.5.

3.3.5 Temporal Consistency

So far we have treated pixels as being independent in space and time. The former could be dealt with using a Markov random field, but at the cost of much slower evaluation. In contrast here we introduce temporal dependence using a recursive Bayes filter:

$$P(M_j^t|\mathbf{Y}^t) = \frac{P(\mathbf{y}^t|M_j^t, \mathbf{Y}^{t-1})P(M_j^t|\mathbf{Y}^{t-1})}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \tag{3.17}$$

where M_j, $j \in \{f, b\}$ is the foreground/background model, $\mathbf{y}^t$ the value of pixel $\mathbf{y}$ at time t and $\mathbf{Y}_t = [\mathbf{y}^t, \mathbf{y}^{t-1}, \ldots]$ the values of pixel $\mathbf{y}$ up to time t.

Following the generative model of Bibby and Reid (2008), we assume conditional independence and write:

$$\begin{aligned} P(M_j^t|\mathbf{Y}^t) &= \frac{P(\mathbf{y}^t|M_j^t)P(M_j^t|\mathbf{Y}^{t-1})}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} = \\ &= \frac{P(\mathbf{y}^t|M_j^t)\sum_{i\in\{f,b\}}\left(P(M_j^t|M_i^{t-1})P(M_i^{t-1}|\mathbf{Y}^{t-1})\right)}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \end{aligned} \tag{3.18}$$

where

$$P(\mathbf{y}^t|\mathbf{Y}^{t-1}) = \sum_{i\in\{f,b\}} P(\mathbf{y}^t|M_i^t)P(M_i^t|\mathbf{Y}^{t-1}) \tag{3.19}$$

The recursion is initialised by $P(M^0) = P(M_j)$.

To estimate $P(M_j^t|M_i^{t-1})$ for $i \in \{f, b\}$ and $j \in \{f, b\}$ we write:

$$P(M_j^t|M_i^{t-1}) = \frac{\zeta_{ji}}{\zeta} \tag{3.20}$$

where ζ is the total number of pixels in the image and ζ_{ji}, $i \in \{f, b\}$ and $j \in \{f, b\}$ is the number of pixels that were of type i at the previous frame and are of type j at the current frame. For example a pixel is considered foreground (i.e. of type f) at time t if $P(\mathbf{y}^t|M_f^t) > P(\mathbf{y}^t|M_b^t)$.

In this work we only enforce first order temporal consistency i.e. $\mathbf{Y}^t = \mathbf{y}^t$, which means that the current value of the foreground or background posterior depends only

on the value at the previous time step.

3.3.6 Online Adaptation

Online adaptation of the foreground and background statistical models is achieved, like in Bibby and Reid (2008), using linear blending with variable learning rates $\alpha_i, i = \{f, b\}$:

$$P^t(\mathbf{y}|M_i) = (1 - \alpha_i)P^{t-1}(\mathbf{y}|M_i) + \alpha_i P^t(\mathbf{y}|M_i) \tag{3.21}$$

where $i = \{f, b\}$.

Unlike Bibby and Reid (2008) however, we use the distance transform as a measure of the uncertainty in the contour: pixels far from the contour and inside the projection will most likely be foreground, while pixels far from the contour and outside the projection will most likely be background. We set a threshold distance and update the foreground and background histograms only where the absolute value of the distance transform is bigger than this threshold. This threshold is equal to the size of the band in which we evaluate energy function (fixed at 16 pixels, in all our experiments). While this might mean that we do not update with some pixels that actually are foreground or background, it does allow us to use higher adaptation rates and ensure more stable tracking. Here we use a foreground learning rate of 5% ($\alpha_f = 0.05$) and a background adaptation rate of 2% ($\alpha_b = 0.02$).

3.4 Implementation and Performance Analysis

In this section we discuss a number of implementational issues and analyse the computational requirements of the algorithm. We use 640×480 images for all experiments.

In the present work the appearance models are represented by RGB histograms using 32 bins per channel i.e. histograms with $32 \times 32 \times 32$ bins. The foreground and background histograms are initialised using the pixels located inside and respectively outside the contour of projection of the known 3D model under the initial approximation of the pose.

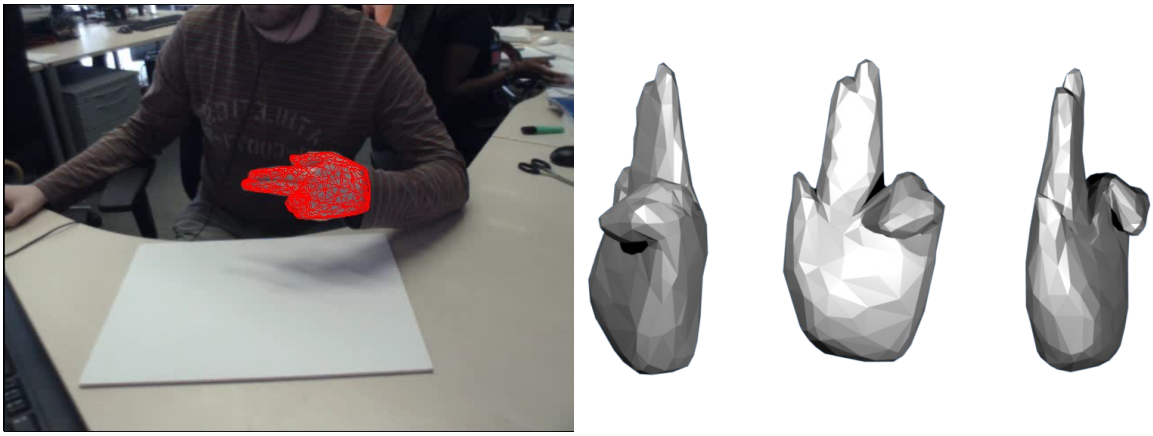


Figure 3.4: The average processing time when using the model on the right with the image on the left is 2.91ms. With this model the algorithm usually required between 15 and 25 iterations for convergence, resulting in an average processing speed of 15-20 frames per second. The resolution of the images is 640×480 and the model has 680 triangles.

Each iteration proceeds as follows:

- Render the 3D model with the current estimate of the pose. The rendering pipeline is similar to that of OpenGL except that we generate *two* depth buffers: one for the closest points in the 3D model (from the camera) and one from the most distant points. To generate the most distant points we use a reversed Z-buffer algorithm i.e. instead of choosing only the pixel with the smallest depth, we choose the one with the biggest depth.

- Compute the contour of the projection and the signed distance transform of this contour.

- Compute the partial derivatives of the signed distance transform.

- Compute the partial derivatives of the energy function.

- In the gradient descent case: make a step change to the pose in the direction of the gradient. The step size may be found using a line search algorithm or be a preset adjustable parameter.

- In the conjugate gradient case: make a step change to the pose in the direction of the gradient and in the direction of the conjugate gradient. Evaluate the

Processing Stage	Time
Rendering the 3D model	0.64 ms
Finding the contour of the 2D projection	0.18 ms
Distance transform of the contour of the 2D projection	1.50 ms
Computation of $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$	0.15 ms
Computation of all $\frac{\partial E}{\partial \lambda_i}$	0.44 ms

Table 3.3: Detailed processing time for one iteration of the minimisation of our energy function, when using gradient descent without the line search (so no evaluation of the energy function is necessary).

energy function in both cases and choose the minimum. The step size may also be found using a line search algorithm or be a preset adjustable parameter.

Most of the tasks involved in one iteration of our algorithm operate per-pixel, so we can achieve significant gains by exploiting the high degree of parallelism. We use the CUDA framework (NVIDIA, 2009) to implement various steps on a GPU. Average processing time per iteration (when using gradient descent) lies between *2.5* and *5ms* using a Geforce GTX 285 video card, for a 640×480 RGB image, depending on the size of the 3D rendering and the complexity of the 3D model. The more pixels being rendered, the bigger the processing time. However, since the energy function is only evaluated in a band around the contour, an increase in image size will not affect the processing time. For example, for the model shown in Figure 3.4, the average processing time per iteration is *2.91ms*. Table 3.3 summarises the average amount of time taken by each stage of the evaluation, for a single iteration of the gradient descent. Since the processing is done by the GPU, the host CPU is free to do other tasks during processing.

3D Rendering Engine

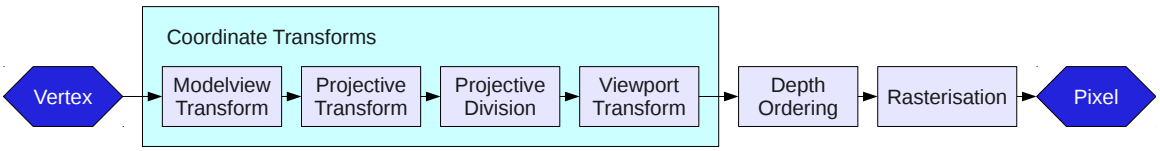


Figure 3.5: 3D Rendering Pipeline

For every iteration of our algorithm, the 3D model must be rendered with a given pose and projection matrix. The distance transform needs to be computed only in the region of the image where the object is located. When a standard rendering engine is used, this region of interest must be computed after the rendering. Also, to our knowledge, it is not straightforward for OpenGL or DirectX to render multiple Z-buffers (requiring multiple separate renderings). We chose to implement our own 3D rendering engine so the region of interest is computed at *render time*. We can also easily add support for multiple Z-buffers. Finally, since there is no need to consider lighting or texturing (which would have been costly in a CPU implementation), our custom rendering engine operates at a similar speed to OpenGL or DirectX (a difference of less than 1ms, in our tests).

Figure 3.5 shows the steps undertaken by our rendering engine to draw a single image pixel, from a known 3D point. Just like OpenGL, first the 3D point must be converted from the object coordinate system to the camera coordinate system, using Equation 3.11. In the second and third steps Equation 3.9 is used to obtain the *normalised device coordinates* for the x and y coordinates of the pixels. The normalised z coordinates have the following expression:

$$z_{NDC} = \frac{f+n}{f-n} + \frac{2fn}{f-n}\frac{1}{Z} \tag{3.22}$$

where f and n are the distances with respect to the far and near planes, respectively. We use $f = 400$ and $n = 1$. In this coordinate system the x, y and coordinates are mapped to the range (-1, 1). This is a non-linear mapping, meaning that closer points have higher resolution than more distant points.

In the viewport transform the 3D coordinates are mapped into the 2D area of the rendering window.

We assume the 3D model to be a triangle mesh. We use interpolation to obtain the x, y and z values for the points inside the triangles. The Z-buffer algorithm is used twice for each pixel in the image, to select the z value for both the closest and the most distant 3D point which project to that 2D pixel.

In the multi-object case multiple renderings are generated: one for each object separately and one with all the objects together – the depth map.

We implemented both CPU and GPU versions of the rendering engine. In the GPU case we use a separate thread for each triangle in the model, while the CPU version uses separate threads for each type of image (wireframe and filled triangles). For the GPU Z-buffer implementation we use global memory atomic write operations to avoid any possible read-write-modify hazards. Note that it is also possible to use one thread per pixel, rather than one thread per triangle. This would mean that each thread would have to check its membership to each triangle, which would quickly become prohibitively expensive as the number of triangles increases. Preprocessing steps could be used to establish which pixels belong to which triangles, but these would also take extra processing time.

In most cases, the GPU implementation is much faster than the CPU implementation. It also scales much better with the number of triangles in the mesh. For example, the model in Figure 3.7 (which has 8886 triangles), was rendered in 5.6 ms with the CPU renderer and in 2ms with the GPU renderer. The only case where the CPU implementation is faster is when the model has few triangles (less than 100) but lots of pixels, i.e. its projection is big (more than 100x100). In this case there are considerably more atomic operations than threads, so the parallelisation is ineffective. Our current implementation chooses dynamically between the two rendering engines. This accounts for the difference in processing time per iteration between that reported in Table 3.2 and the one from Table 3.3, because in the experiments which led to the results shown in Table 3.2 the algorithm used the CPU renderer, while for the other ones, the algorithm used the GPU renderer.

GPU Contour Extraction

We compute the contour of the projection by convolving the projection image with Scharr kernels i.e. by using a standard edge detection algorithm in the projection image. The Scharr (2000) kernels are:

$$\text{Horizontal:} \quad \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \qquad \text{Vertical:} \quad \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

This implementation would have been impractical using a sequential algorithm, but it is very parallelisable. Our parallel implementation uses one thread per pixel. In the multi-object case (where a rendering of each object separately is known) the contour is extracted for each object separately.

GPU Distance Transform

To compute the distance transform we use an algorithm similar to Felzenszwalb and Huttenlocher (2004). For each location $[x, y]$ in a binary image I, the 2D distance transform $DT(x, y)$ is computed as:

$$DT(x, y) = \min_{[x', y'] \in \Omega} \left((x - x')^2 + (y - y')^2 + I(x, y) \right) \tag{3.23}$$

This is done by splitting the computation into two passes, first by columns (computing per column 1D distance transforms) and then by rows, by writing:

$$DT(x, y) = \min_{x'} \left((x - x')^2 + DT'(x, y) \right) \tag{3.24}$$

where $DT'(x, y)$ is a set of per column 1D distance transforms of I:

$$DT'(y) = min_{y'} \left((y - y')^2 + I(x, y) \right) \tag{3.25}$$

Each 1D distance transform is computed by finding the lower envelope of the per column parabolas defined by Equation 3.25.

Our parallel implementation uses one thread for each column and then one thread for each row in the image. The second pass does not take full advantage of the CUDA framework capabilities because memory access is not coalesced.

In the multi-object case a distance transform is computed for each object.

GPU Finite Differences and Energy Function Evaluation

The computation of the partial derivatives of the signed distance transform uses central finite differences, parallelised by using one thread per pixel. The partial derivatives of the energy function are also computed by using one thread per pixel and then transferred back to host memory.

3.5 Results

We tested our method on a variety of models and video sequences. In this section we demonstrate some of the qualities of our method, such as robustness to initialisation, motion blur and occlusions. We also demonstrate the advantage of including the most distant points of the object in the evaluation of the energy function. We showcase the difference between using the pixel-wise posteriors formulation of the level set framework and the classical log likelihood formulation. We compare our method with the one of Brox et al. (2009), which is at or close to the current state of the art in 3D pose tracking. Although Brox et al. (2009) combine region-based tracking with optical flow and SIFT features while we only use the silhouette, we achieve similar or better results, but with a processing time that is hundreds of times smaller. Finally we compare the poses recovered by our method for each frame of a video with the ones obtained by a SLAM (Simultaneous Location and Mapping) system, over the same video.



Figure 3.6: Typical algorithm run for one frame. left - initialisation, middle - intermediate iteration, right - final result.

Figure 3.6 shows an example of our method performing segmentation on a single

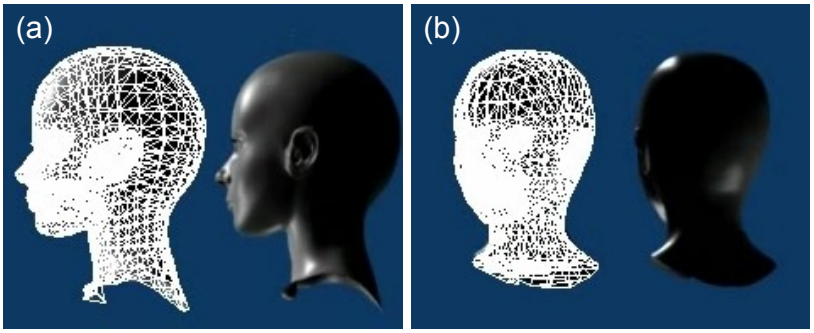


Figure 3.7: Failure due to the symmetry of the object. (a) – left: recovered pose, right: rendered frame, (b) – left: recovered pose, right: rendered frame.

frame [1]. Here we are able to successfully recover translational changes in x and y of up to ±40% of the object size, a similar amount in z (scale) and rotational changes of 50 degrees on the x and y axis and 70 degrees on the z axis. Depending on the model and image data, we were able to recover rotations up to 90 degrees on each axis. With regard to translation, the algorithm usually converges as long as the projection of the 3D model intersects the object in the image (also depending on the model and of course the image data itself). When the projection overlaps with the object in the image, even by a small amount, a pressure develops in the energy function with the effect of increasing this overlap, effectively pulling the object in the right direction.

Figure 3.7 shows a failure scenario for our method. It is an artificially generated video sequence in which the camera is rotated around a human head. The segmentation is correct but the pose is wrong. This problem arises because the object is symmetrical; the contour alone is insufficient to determine the pose unambiguously. This is of course a limitation of all pose recovery algorithms based solely on the occluding contour.

In standard edge based tracking, because of motion blur, edges often disappear, or have unstable positions. This leads to an advantage of a region-based tracker over an edge based tracker: its robustness to motion blur. In Figure 3.8 we use a soft-drink can and 3D model to showcase a typical scenario when our method is able to successfully track the 3D object while any local feature based pose recovery algorithm

[1]The 3D model of the space-shuttle was obtained from http://www.nasa.gov/.



Figure 3.8: Robustness to motion blur. Left – original image, Right – recovered position.

would have most likely failed.

Figure 3.9 shows the effect of imposing temporal consistency on the posterior probabilities. We plot the per pixel difference $P_f - P_b$, where $P_f - P_b > 0$, scaled to $[0, 255]$. Brighter pixels correspond to a larger difference between P_f and P_b. When imposing temporal consistency, the separation between the two regions is better, which leads to a faster and more accurate pose recovery.

A potential source of problems when using strong prior 3D (or 2D) shape knowledge in tracking and pose recovery are discrepancies between the model and the real target. Figure 3.10 shows frames from a sequence where we track a human hand with an approximate model. Tracking is still accurate because our method is region-based and probabilistic.

Rapid motions produce big differences in pose between successive frames. Because of the large basin of convergence our method can often recover the pose even in these cases. The algorithm does have slow convergence when surfaces appear and disappear so recovery of rapid motions in these cases is more difficult. Note that we do not use a motion model, though this could easily be incorporated. This would greatly help in the prediction of the appearance and disappearance of surfaces.

The evolution of the contour of the projection is not always dictated by the visible part of the object. Surfaces may appear and may disappear. As mentioned in Section 3.3, we also take into consideration the most distant points which project to the

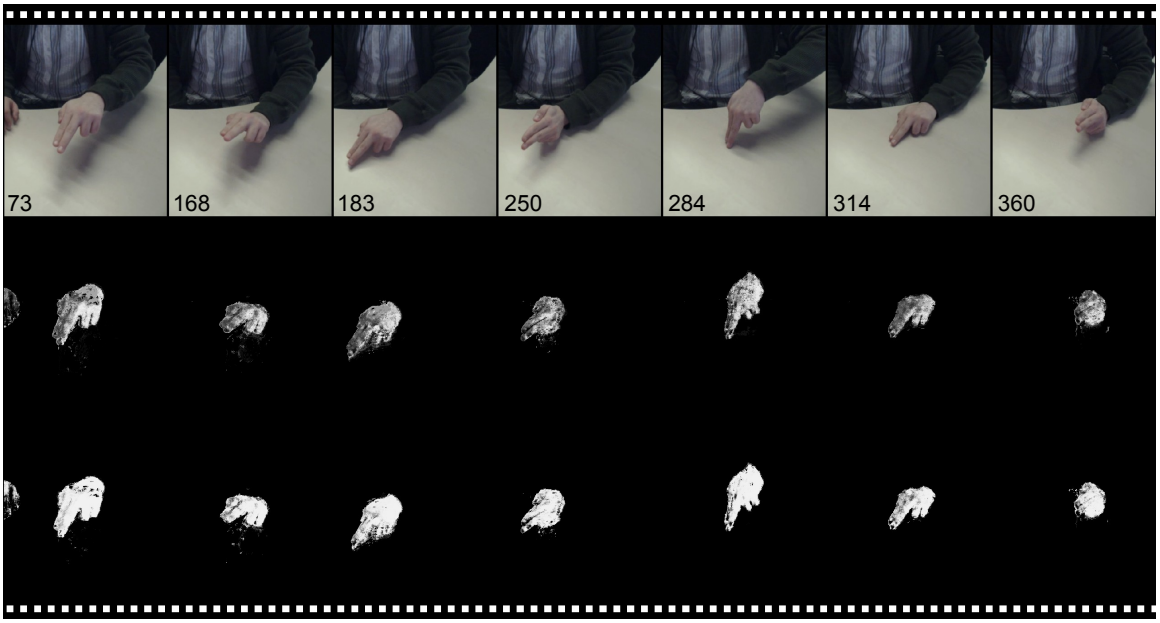


Figure 3.9: Effect of temporal consistency on the posterior probabilities. The top row shows the original images. In the second and third row, we plot the per pixel difference $P_f - P_b$ where $P_f - P_b > 0$, scaled to $[0, 255]$. The separation between the foreground and background regions is better when imposing temporal consistency (third row).

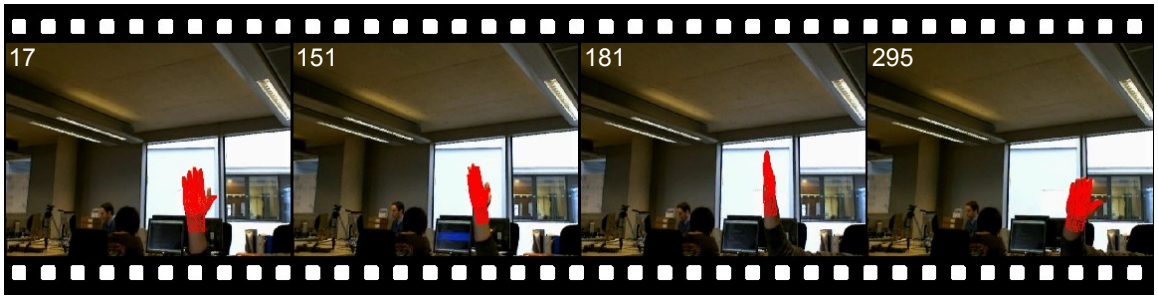


Figure 3.10: Film strip showing our method tracking a hand. Our algorithm can track the hand in this sequence at around 20fps.

contour of the projection, in the energy function. Figure 3.11 shows excerpts of a video sequence where the camera rotates above a soft-drink can. The view of the can moves to directly above, with the side obscured, back to a view with the side visible again. When allowing only the closest points to influence the contour evolution, the pose incorrectly flips. In contrast the behaviour is correct when the furthest points are also considered.

We use our video sequences to perform a critical performance analysis comparing

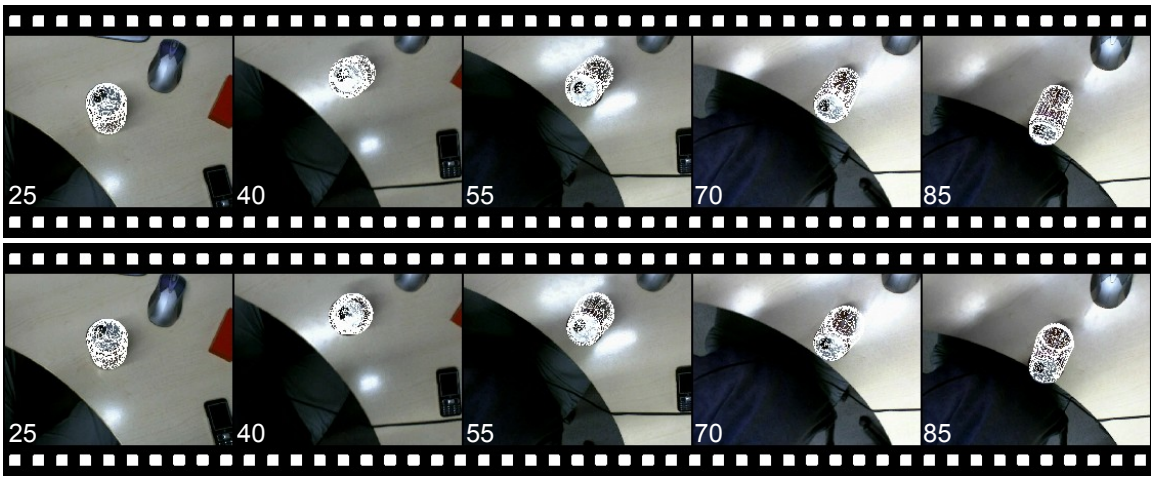


Figure 3.11: Film strip showing 5 frames from a video sequence in which the camera moves above a soft-drink can in a circular motion. Between frames 25 and 55 the view of the can moves to directly above, with the side obscured, back to a view with the side visible again. When allowing only the closest points to influence the contour evolution, the pose incorrectly flips (above). In contrast the behaviour is correct when the furthest points are also considered (below).

the performance of both the pixel-wise posteriors energy function that we use in this chapter and the log likelihood one (like that of Cremers et al. (2007) and Dambreville et al. (2008b)), for 3D tracking. For each frame we compute 120 perturbations around the correct pose (20 for each pose parameter), in total more than 200,000 experiments. We measure whether each algorithm converges and how many iterations are required for convergence to the true pose [2], with success defined by a difference of at most 5 degrees in rotation and 5% of object size in translation. Figure 3.12 summarises the results. Though there is little choice between the objective functions for recovery of translation in x and y axis (i.e. parallel to the image plane), the pixel-wise posteriors energy function outperforms the log likelihood objective function for recovering translation in the z axis and for all 3 rotation axes. Finally, the number of iterations necessary for convergence (when convergence was possible) is considerably higher when using log likelihood (for both rotation and translation). For these experiments we use cuboid objects, like the one in Figure 3.3.

In Figure 3.13 we depict a film strip showing the behaviour of PWP3D when

[2]Without access to the ground truth, the true pose was measured subjectively

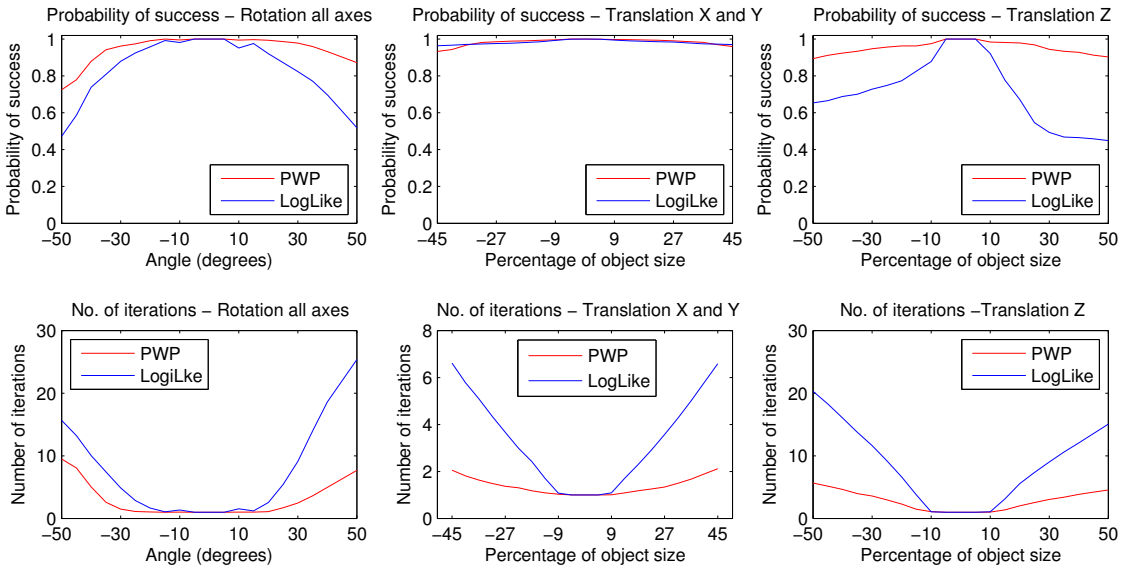


Figure 3.12: Performance analysis: Probability of convergence to the correct pose generated from over 200,000 experiments.

subjected to high degrees of occlusion and noise. When both the top and bottom parts of the soft-drink can are visible, the pose is correct, even though most of the interior part of the object is occluded. When the bottom part of the can becomes occluded the pose becomes incorrect. The object moves "up" in the image because there is no longer a reason for it to stay "down". The algorithm converges back to the correct position once the bottom part stops being occluded. Throughout our testing PWP3D has consistently been able to recover from an incorrect pose after no more than 10 frames.

We compared our method with the one proposed by Brox et al. (2009), which is at or close to the current state of the art in 3D model based tracking. The authors use three cues for point correspondences: region-based, dense optical flow-based and SIFT-based. In Figure 3.14 we show the results obtained by our method (above) and the one of Brox et al. (2009) (below). The results are very similar, even though we track only the silhouette. Our algorithm is also considerably faster (we can track this object at around 100ms per frame while the authors report processing a frame in about 80s). It is important to note that here we use a single view (above, first row), and reproject in the other (above, second row), while Brox et al. (2009) use

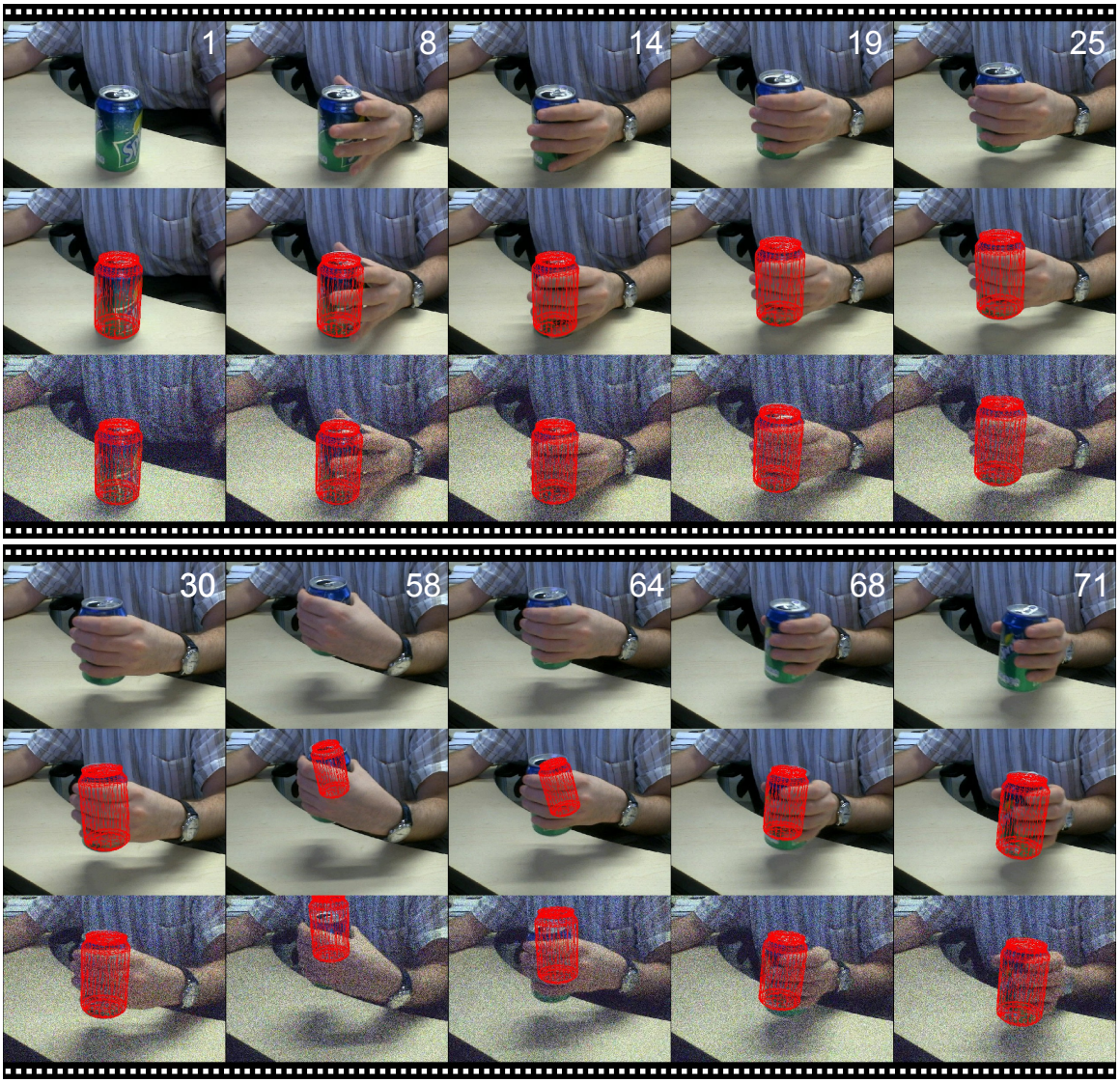


Figure 3.13: Film strip consisting of 10 frames from a video which showcases the PWP3D behaviour when subjected to occlusions and high levels of noise. For the first 6 frames (when both the top and bottom parts of the soft-drink can are visible) the pose is correct, even though most of the interior part of the object is occluded. When the bottom part of the can becomes occluded the pose becomes incorrect. The object moves "up" in the image because there is no longer a reason for it to stay "down". The algorithm converges back to the correct position once the bottom part stops being occluded.

three views (from which we included 2 in Figure 3.14). Brox et al. (2009) will of course achieve better results when the pose is more ambiguous (i.e. when more poses project to the same silhouette), but this experiment still demonstrates the efficacy of our method.

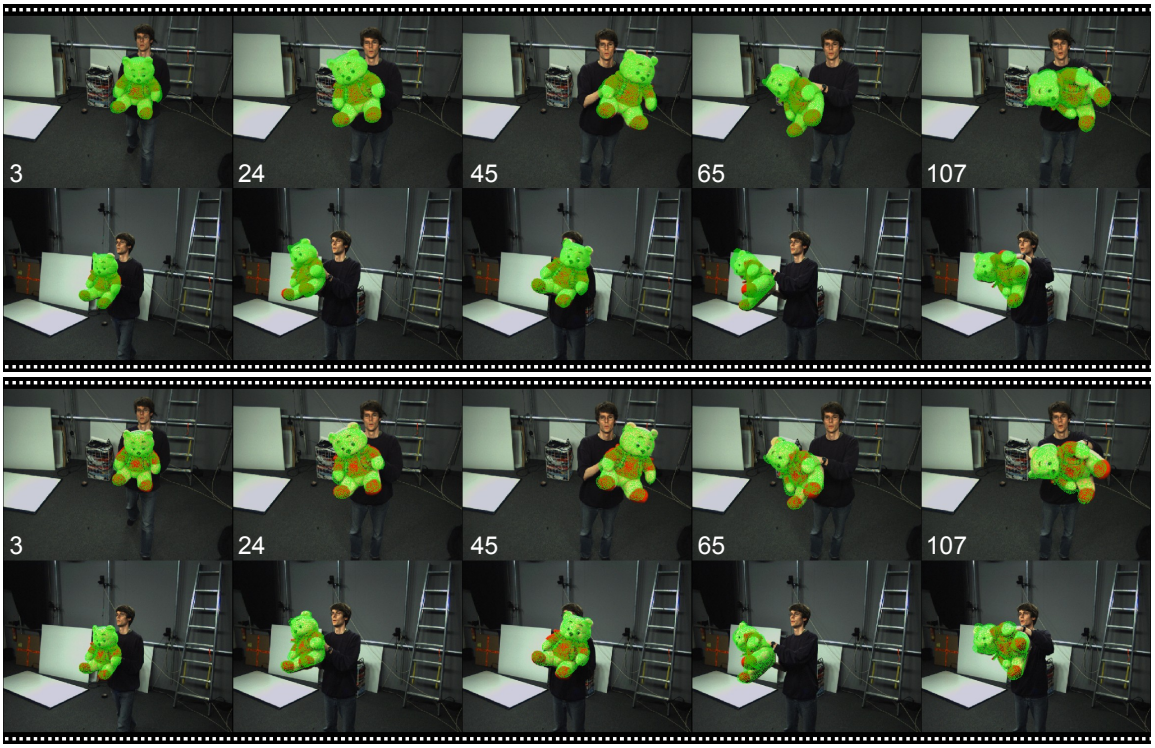


Figure 3.14: Comparison between our method (above) the one presented by Brox et al. (2009). Although Brox et al. (2009) combine region-based tracking with optical flow and SIFT features to achieve tracking, while we use only the silhouette, our results are very similar, if not better. Note that here we use a single view (above, first row), and reproject in the other (above, second row), while Brox et al. (2009) use three views (from which we included 2 here).

Our system obtains the pose parameters for a 3D object relative to a fixed camera. This is the same as obtaining the pose of a 3D camera relative to a stationary object. SLAM systems build a sparse 3D map of the environment and calculate the pose of a 3D camera relative to that map. Therefore we can obtain a measure of accuracy for PWP3D by comparing it against a SLAM system. Neither SLAM nor PWP3D provide the exact pose of the camera but the two systems should converge to very similar results. In Figure 3.16 we depict the pose parameters obtained by our system and those obtained by the PTAM (Parallel Tracking and Mapping) system of Klein and Murray (2007), which is at or close to the current state of the art in structure from motion. A few frames from the video used by us to compare our system with PTAM are shown in Figure 3.15. The two are very close except for two regions in

rotation on X and Z (around frame 350 and 460) and one on translation in Z (around frame 500). Note that SLAM/PTAM uses many features to obtain the pose of the camera whereas PWP3D uses just the silhouette. The differences in rotation are caused by the fact that we are tracking only the silhouette and, in those regions, the object was viewed from the side. A tilt in the X or Z direction had no effect on the silhouette. The difference in translation on Z is caused by a high degree of motion blur, which corrupted the features used by PTAM.

In Figure 3.17 our algorithm uses two views to disambiguate the pose of a hand. The top two rows show the two views. The straight on view (top row) of the hand is ambiguous i.e. multiple poses can generate the same silhouette. Row three shows the pose obtained by our algorithm when processing only the first view (top row). The algorithm is confused by the ambiguity in the pose-silhouette function and converges to a wrong result. When adding the second view (second row) the pose-silhouette function is disambiguated and the pose is recovered successfully in both views (last two rows). Processing time suffers considerably, as processing two views is twice as slow as using a single view. Multiple threads and/or video cards (one for each view) could be used to speed up the algorithm.

Figure 3.18 shows our algorithm tracking two objects, a soft drink can and a hand. In the top film strip the hand occludes the soft drink can, while in the second film strip the soft-drink hand occludes the hand. Tracking is consistent in both cases. Processing time does again suffer considerably. Tracking two objects is about 2.5 times slower than tracking a single object, two times slower because two energy functions are computed, plus the extra processing required for generating and using the depth map.

3.6 Conclusions

In this chapter we have described a novel model-based 3D tracker, constructed on the assumption that the segmentation of an object in an image is fully defined by the 6 DoF pose space of its known 3D model. The method is based on the computation of

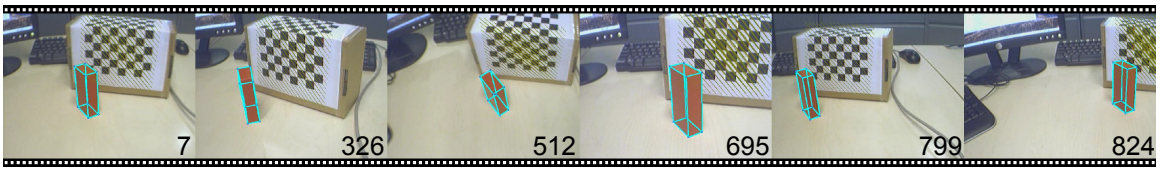


Figure 3.15: Filmstrip showing 6 frames from our PTAM comparison video. We used a checker board pattern to calibrate our results with the PTAM (Klein and Murray, 2007) ones.

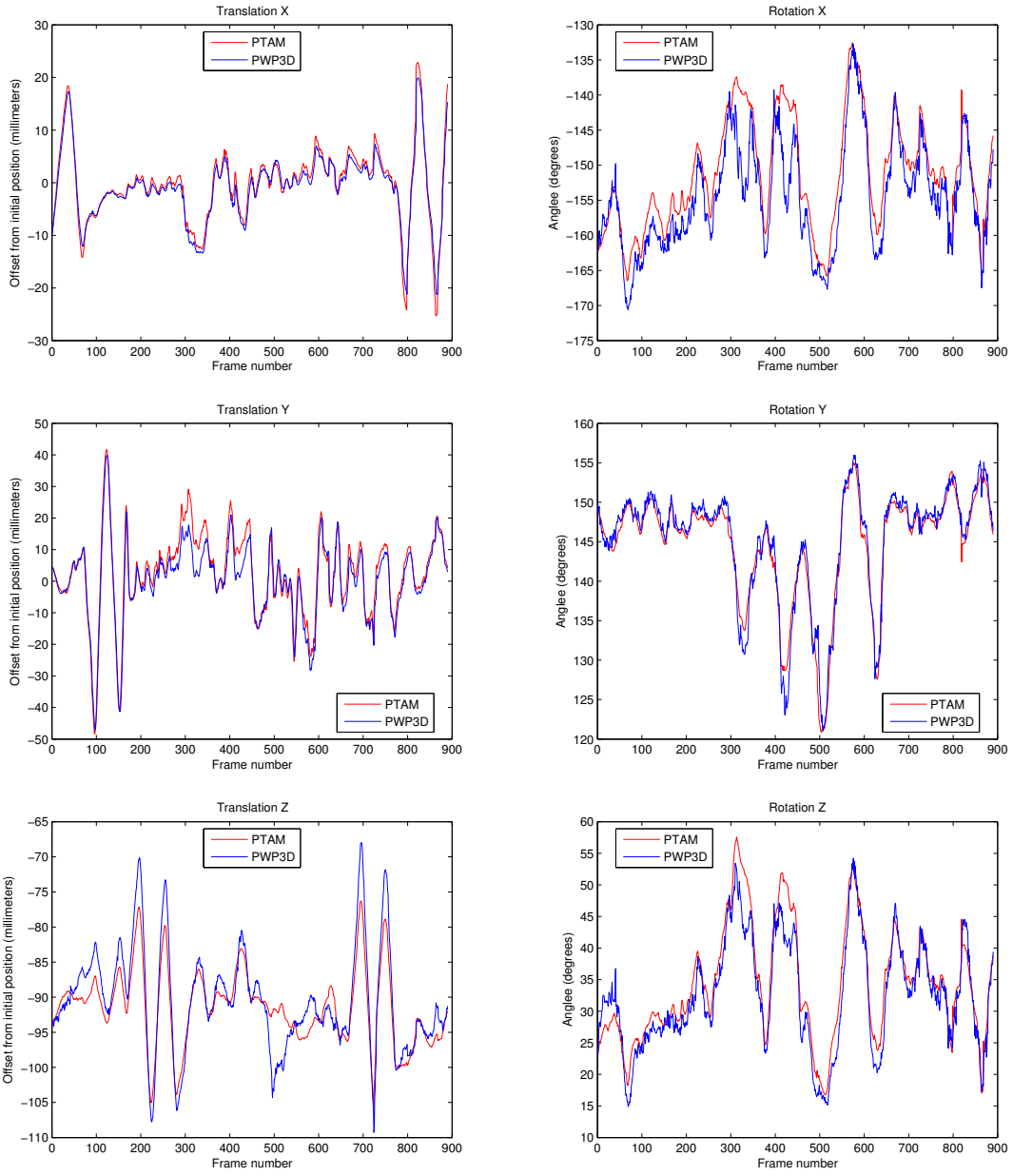


Figure 3.16: Comparison between camera pose output when using PTAM (Klein and Murray, 2007) and PWP3D.



Figure 3.17: Multi view tracking. The top two rows show the two views. When processing only the first view (top row) the results are incorrect (third row) because the silhouette to pose mapping is ambiguous (multiple silhouettes can be generated by the same pose). When using two views the ambiguities are eliminated and the results are correct for both views (bottom two rows).

the derivatives of a level set based segmentation energy function with respect to the pose parameters of the known 3D model. Segmentation and 3D rigid pose are then recovered by standard nonlinear optimisation techniques, such as gradient descent and conjugate gradients, resulting in a fully variational, probabilistic, one shot, level set minimisation. Inspired by Bibby and Reid (2008) we have adopted the use of pixel-wise posteriors (in contrast to likelihoods) and shown that, as in the 2D case addressed by Bibby and Reid (2008), we achieve performance gains also in the 3D case. In addition, we have introduced further improvements at the pixel level such as consideration of temporal consistency and improved online adaptation of foreground

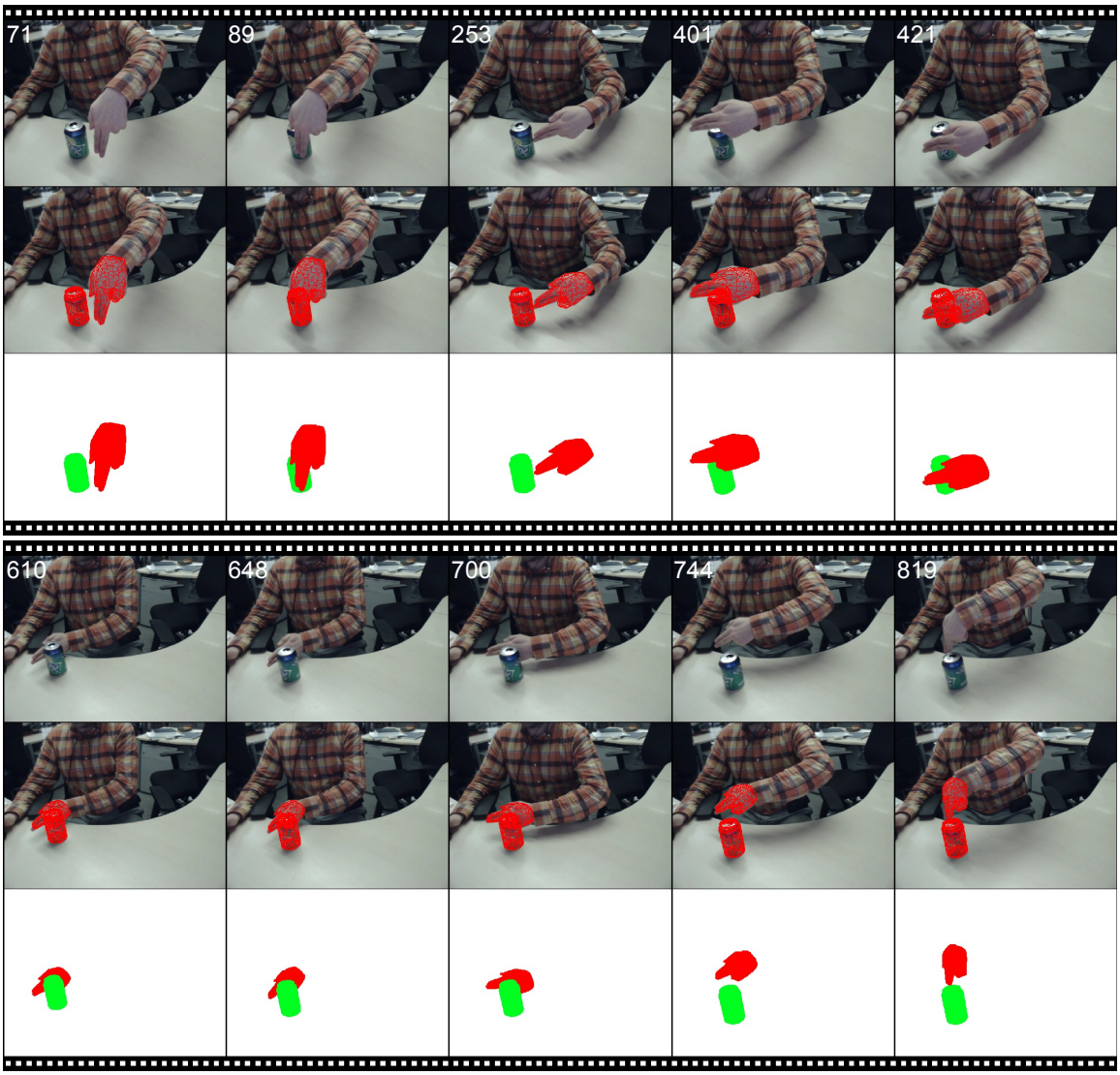


Figure 3.18: Our algorithm tracking two objects, a soft drink can and a hand. The top film strip shows the hand occluding the soft drink can. The bottom film strip shows the opposite, the soft drink can occluding the hand. The third row in each film strip shows the resulting depth map.

and background appearance models. Importantly, we have also demonstrated that, because of its highly parallel nature, our algorithm is amenable to real-time implementation using GPU acceleration, making this one of the few model and region based 3D trackers to achieve real time performance. Our method shares the benefits of region based segmentation with shape priors, such as robustness to occlusion, noise or motion-blur, while still allowing for large changes in rotation and translation to

be successfully recovered, and these aspects have been experimentally demonstrated. We have shown how straightforward extensions lead to the ability to track multiple objects from multiple views, which overcomes ambiguities created by the fact that we only use silhouette information in finding the 6D pose.

A few issues remain. Tracking can be lost when there is no overlap between the positions of the object at consecutive frames or when the object moves throughout ambiguous poses. Application specific solutions to these problems are detailed in Chapter 4. Furthermore, the PWP3D algorithm can only track rigid objects. It would impractical to extend it track full articulation or fully deformable objects, because of the high dimensionality of the space that would need to be searched. Several techniques exist for learning manifolds of 2D shapes by use of dimensionality reduction techniques, and in Chapter 5 we show how such techniques could be used to learn manifolds of both 2D and 3D shapes and how a tracking algorithm that uses such manifolds could be developed.

Chapter 4

PWP3D Applications

In this chapter we show two example applications of our 3D tracker: a driver assistance system that links our tracker with a single view traffic sign detector and a 3D hand tracker that links our 3D generic object tracker with data coming from an off-the-shelf accelerometer and classification results coming from a multi-class SVM classifier. Details about driver assistance system were published in Prisacariu et al. (2010) and Timofte et al. (2011) and the about the hand tracker in Prisacariu and Reid (2011b) and Prisacariu and Reid (2012a).

4.1 Introduction

In the previous chapter we showed several qualitative and quantitative examples where our PWP3D tracker produces good results. In this chapter we detail two complete end-to-end applications of our tracker, a driver assistance system based on a traffic sign tracker in Section 4.2 and a human computer interaction system based on a hand tracker in Section 4.3, with conclusions for both applications being presented in Section 4.4. These applications also serve as examples of how two of the problems mentioned in Chapter 3 can be solved. First, if there is no overlap between consecutive frames, the 3D tracker often loses tracking. In the driver assistance example application we solve this problem by augmenting the tracker with a (traffic

sign) detector. Second, the same silhouette can correspond to multiple different poses. In the HCI example application we use an accelerometer placed on the hand to help disambiguate between these kind of poses. Third, PWP3D can only work with rigid objects. Also in the HCI example, we use a classifier to select between a set of activity specific articulated hand configurations.

4.2 Driver Assistance

Traffic signs are designed to help drivers reach their destination safely, by providing them with useful information. However, when they are missed or misunderstood accidents often happen. A recent statistic shows that over 98% of car accidents happen because the driver was distracted. By attracting the driver's attention to the road signs many accidents would be averted. As a result there has been much work towards a fast and reliable traffic sign detection and tracking system.

Most current work involves combining a detector with a Kalman filter, like in Fang et al. (2003) or Ruta et al. (2010), or with a particle filter, like in Meuter et al. (2008) or Lopez and Fuentes (2007). Such methods rely on a predictable car motion model or reliable feature detectors. For example, Piccioli et al. (1994) and Fang et al. (2003) assume the car moves in a straight line and with constant velocity. As feature descriptors, trackers usually use edges or some kind of information extracted from the traffic sign shape. Fang et al. (2003) explicitly model geometric properties of the traffic sign shape (i.e. a triangle shaped sign has to be equilateral). This leads to a lack of robustness when subjected to occlusions, deformations or motion blur. Meuter et al. (2008) track circular signs and assume these have a coloured border on a white interior and that clear edges can be extracted. This approach would not scale to differently shaped signs and is again vulnerable to motion blur. A solution to some of these problems is region based tracking. Regions are more stable than edges so tracking is more robust, and they are less affected by occlusions or motion blur.

To our knowledge, with notable exceptions like Meuter et al. (2008) and Timofte et al. (2009), most of previous road sign work was *2D*. That is the position of the

traffic sign was tracked in the *image* rather than in *3D space*. Meuter et al. (2008) use inertial sensors mounted on the car to help them obtain an approximation of the 3D pose of the signs, with respect to the car. Unfortunately this approach would fail when the traffic sign does not point towards the car, like in the case shown in Figure 4.1. Here the no right turn sign (located in the upper left part of the image) does not have the same rotation as the inertial sensor mounted on the car. An alternative method is presented by Timofte et al. (2009), where multiple views, 3D reconstruction, and an Minimum Description Length based method are used. Although a 3D pose is recovered, processing time is very long i.e. it is an offline method for 3D mobile mapping purposes.



Figure 4.1: Importance of determining the traffic sign orientation. The no right turn sign (located in the upper left part of the image) does not point towards the car.

Our approach integrates a single view detection and recognition step with the multi-object, model-based, 3D tracker described in Chapter 3. This has several advantages:

- we are able to obtain the full 3D pose of the traffic signs in the image, accounting for the case in Figure 4.1;
- tracking is region based, making it robust to motion blur and occlusions;

- because our tracker uses a GPU based implementation and processes only a small region in and around the detection, we are able to achieve real time performance.

The remainder of this section is structured as follows: we begin in Section 4.2.1 with an overview of our system. In Section 4.2.2 we detail the single view detection and recognition step and in Section 4.2.3 we explain our non-maximum suppression stage. Detector/tracker integration is detailed in Section 4.2.4 and results are shown in Section 4.2.5.

4.2.1 Overview

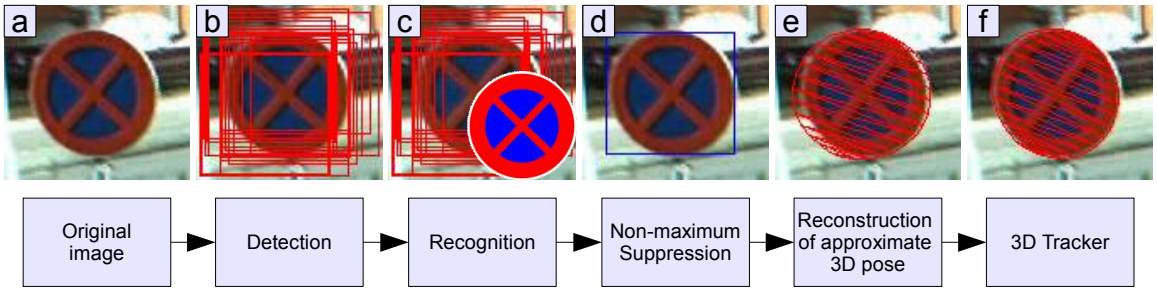


Figure 4.2: Algorithm Overview

Figure 4.2 shows an outline of our algorithm. It consists of five steps. First the signs are detected using a single view detection step. The sign in each detection is then recognised and the best detection for each object is selected. Next, for each sign in the image, an approximate 3D pose is computed, by combining a predicted 3D pose (from the previous frame, with a constant velocity motion model) with a 3D pose obtained from the detection bounding box. The detection bounding box is then converted to a 3D pose using a 4-point planar pose recovery algorithm. Finally, the 3D tracker is iterated from the approximated pose, which results in a refined 3D pose, for each object in the image. We use the object detection and recognition algorithm from Timofte et al. (2009) followed by the PWP3D tracking detailed in Chapter 3.

4.2.2 Object Detection and Recognition

When a new frame is available we first use the object detection and recognition algorithm of Timofte et al. (2009). The single-view detection phase consists of the following steps:

1) Candidate extraction - very fast preprocessing step, where the optimal combination of simple (i.e. computationally cheap), adjustable extraction methods select bounding boxes with possible traffic signs. This step requires an automatic offline learning stage, where an optimal subset of those extraction methods is learnt to yield very few false negatives, while keeping the number of false positives in check.

2) Detection - Extracted candidates are verified further by a binary classifier which filters out remaining background regions. It is based on the Viola and Jones (2001) discrete AdaBoost classifier . Detection is performed by cascades of AdaBoost classifiers, followed by an SVM operating on normalised RGB channels, pyramids of HOGs (Bosch et al., 2007) and AdaBoost-selected Haar-like features.

3) Recognition - A hierarchy of SVM classifiers splits the detections in six basic traffic sign subclasses (triangle-up, triangle-down, circle-blue, circle-red, rectangle and diamond) and then another hierarchy of SVM classifiers (for each subclass) assigns the traffic sign type for the different candidate detections.

For more details on this step, the reader is referred to Timofte et al. (2009).

4.2.3 Non-Maximum Suppression

At this stage several detections might be available for each object, as shown in Figure 4.3. We need to select only a single, best segmentation, process known as non-maximum suppression. There are several methods for doing this, one example being Dalal and Triggs (2005), where mean-shift is used to search in the 3D scale–location space.

On the other hand, we have the advantage of knowing colour models for the foreground and for the background. This means that, for all the detected bounding boxes, we can select the one which best segments the traffic sign from the background.



Figure 4.3: Non-maximum Suppression Example Results

To do this we evaluate the Bibby-Reid energy function described in Chapter 3 for all detected bounding boxes and select the one with the highest colour matching score.

With b as the bounding box we write:

$$P(b) = \prod_{x \in \Omega} \Big(H_e(a(x)) P_f + \big(1 - H_e(a(x))\big) P_b \Big) \tag{4.1}$$

where we replace the level set function Φ with a function $a(x)$ indicating membership of pixel x to the inside or outside of the bounding box:

$$a(x) = \begin{cases} 1 & x \text{ inside the bounding box} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Obviously, when an object is detected for the first time, we have no colour model yet. In this case we choose the detection with the highest SVM score, and initialize the two colour models $P(y_i|M_f)$ and $P(y_i|M_b)$.

4.2.4 Approximate 3D Pose Reconstruction and 3D Tracking

At this stage of the algorithm we have a single bounding box for every traffic sign in the image and we need to track their 3D position (rotation and translation). The core of our tracking procedure is the PWP3D algorithm of Chapter 3.

There are two issues with the above formulation which need to be addressed in order for us to apply this tracker to our problem. The PWP3D tracker needs an initial 3D pose and values for the foreground/background membership probabilities. Also, the PWP3D tracker can lose tracking (because of lack of overlap), so at each new frame, the 2D detections need to be converted to (approximate) 3D poses to be combined with the 3D tracker. These two problems are addressed in the following paragraphs.

Initialization

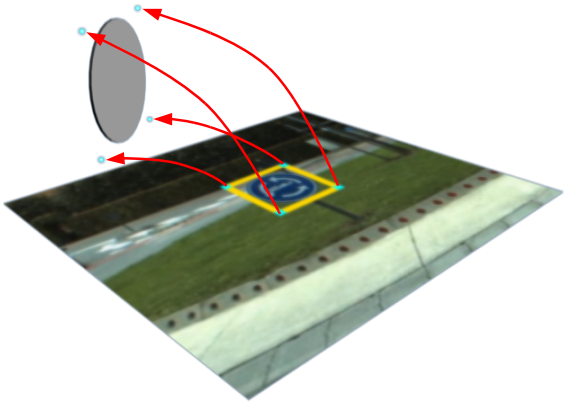


Figure 4.4: 3D–2D point correspondences

Traffic signs can be approximated as planar objects, which means we can use any one of several planar pose recovery algorithms currently available to convert the 2D bounding box surrounding the detection to a 3D pose. We use the currently

state-of-the-art algorithm introduced by Lu et al. (2000).

This algorithm requires (at least) 4 3D–2D point correspondences and works iteratively, minimizing a sum of squared object-space collinearity errors. We use the 4 2D corners of the 2D detection bounding box and relate them to the 4 corners of the bounding box enclosing the 3D model of object, as depicted in Figure 4.4.

An example result is depicted in Figure 4.2e.

Detector–Tracker Integration

The PWP3D tracker will often lose its track if there is no overlap between the positions of the tracked object at consecutive frames. We therefore use the 3D poses recovered from the detection bounding boxes (as presented in the previous section), combined with a constant velocity motion model, to re-localize the 3D tracker.

One common solution is to use a Kalman filter for a purely statistical fusion of the tracking data with the approximate poses from the detector. A Kalman filter represents all measurements, system state and noise as multivariate Gaussian distributions. In contrast, in this work we first compute an approximate 3D pose for each 3D sign at the current frame, by combining the detection with a motion model. This serves as an initialization for the PWP3D pose optimization described above. By iterating the tracker rather than doing a purely statistical data fusion we make no pretence on the type of these probability distributions.

We define a constant velocity motion model:

$$v^{i}_{t_k} = t_{k-1} - t_{k-2} \qquad v^{i}_{r_k} = r_{k-1} r^{-1}_{k-2} \tag{4.3}$$

where k is the current frame, $k-1$ is the previous frame, t is the translation and r is the rotation quaternion from the tracker.

There is also a velocity given by the 2D detections:

$$v^{ii}_{t_k} = u_k - u_{k-1} \qquad v^{ii}_{r_k} = p_k p^{-1}_{k-1} \tag{4.4}$$

where u is the translation and p is the rotation quaternion, obtained from the detector by using the 4 point planar pose recovery algorithm.

When the object is initially detected it is most likely far from the camera, so its rotational motion is less predictable, making the chance of lack of overlap higher. In this case detections should be trusted more. When the object is close to the camera, rotational motion is more predictable so the chance for lack of overlap is smaller. In this case the motion model should be trusted more.

The predicted pose for the current frame becomes:

$$t_k = t_{k-1} + \alpha v^i_{t_k} + \beta v^{ii}_{t_k} \qquad r_k = r_{k-1} q_\alpha v^i_{r_k} q_\beta v^{ii}_{r_k} \tag{4.5}$$

where k is the current frame, $k-1$ is the previous frame, t is the translation from the tracker and u is the translation from the detector. The variables α, β, q_α and q_β are dependent on the distance between the object and the camera. α and q_α are inversely proportional to this distance while β and q_β are proportional to it.

Finally the predicted pose (t_k, r_k) is refined using the tracker.

4.2.5 Results

We tested our algorithm with a multitude of traffic signs types and shapes. In the section we show our system correctly tracking single or multiple traffic signs, of various shapes (rectangular, circular and triangular) and types (pedestrian crossing, parking, give way, etc.). We also show the benefit of using the 3D tracker, instead of just the standard 4 point planar pose recovery algorithm.

We compared the performance of our system by tracking a single sign over a distance of 60m (or 60 frames at 36km/h), with and without the 3D tracker. The car was moving along a straight line, with constant speed. Translation should therefore change linearly, while rotation should remain constant. Figure 4.5 shows the results for a circular sign, Figure 4.6 for a rectangular sign and Figure 4.7 for a triangular sign. At higher distances the object shrinks to only a few tens of pixels, so detection and tracking are prone to errors. Still, the 3D tracker is able to provide smoother

values for the translation. Though there may be little difference between using the 3D tracker and using just the 4 point planar pose recovery algorithm with regard to the translation, the tracker must be used to reliably obtain the rotation.

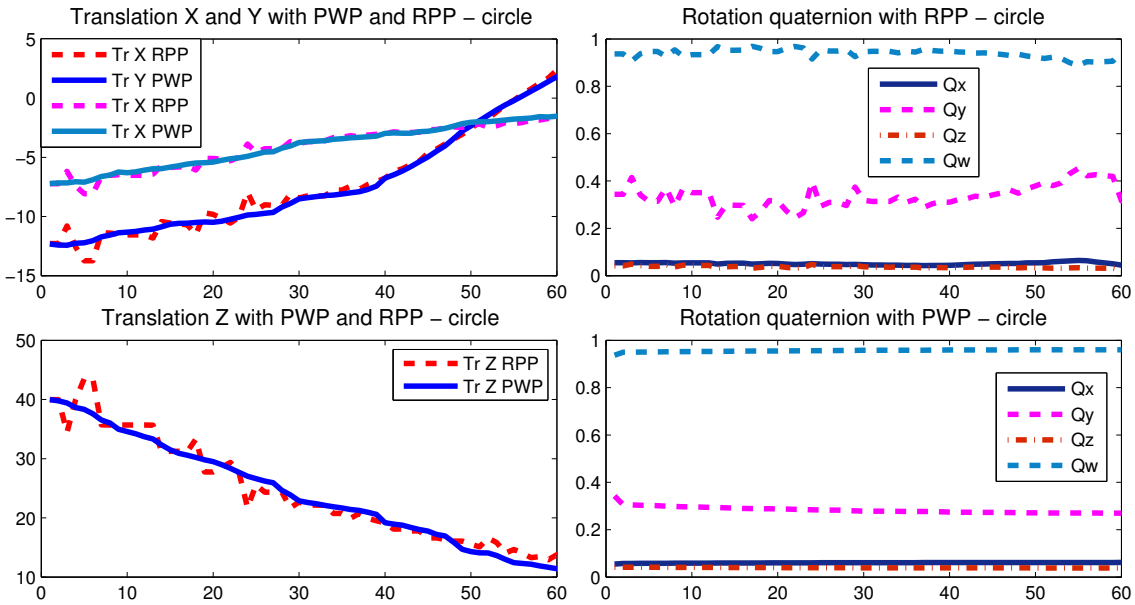


Figure 4.5: System performance while tracking a circle shaped sign over 60m, with just the 4 point pose recovery (RPP) and with the tracker (PWP).

In Figure 4.8, where we show our system tracking a pedestrian crossing sign and obtaining an accurate pose even when the object is far from the camera. Similarly, in Figure 4.9 we show our algorithm successfully tracking a give way sign. An application which comes to mind is to calculate the distance between each sign and the car and to then use it to automatically adjust the speed of the car. Figure 4.10 shows our system tracking multiple round traffic signs. Currently we estimate each pose independently, though of course relative to the camera each sign undergoes the same rigid transformation. This coupling would results in improved performance.

As mentioned several times in this and the previous chapters, our system is able to deal with occlusions. Figure 4.11 shows our system successfully tracking a parking sign, in spite of it being occluded by a tree.

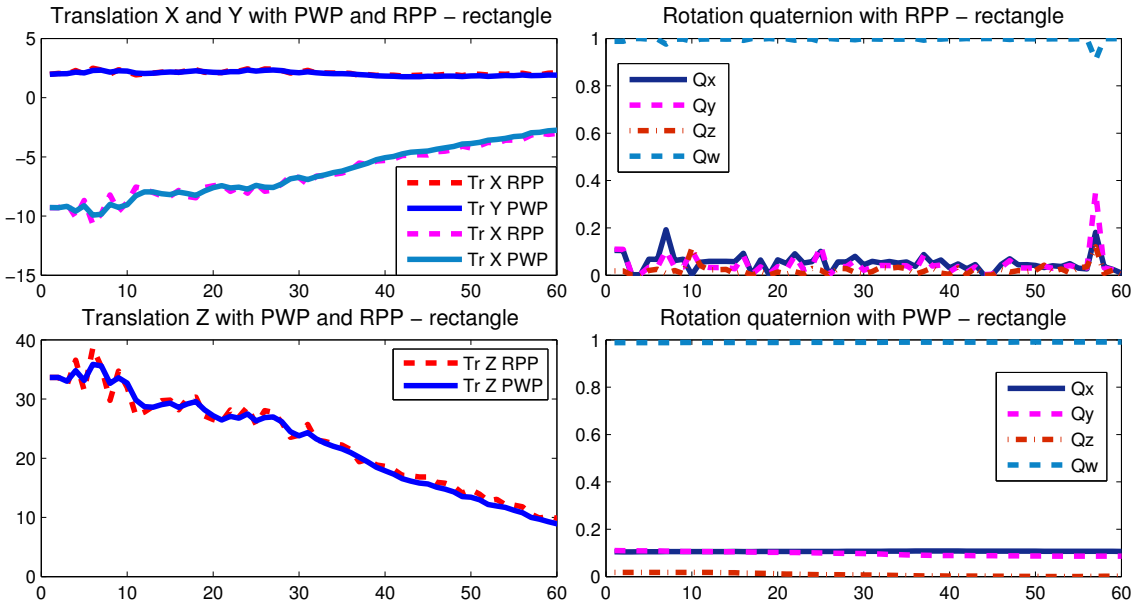


Figure 4.6: System performance while tracking a rectangle shaped sign over 60m, with just the 4 point pose recovery (RPP) and with the tracker (PWP).

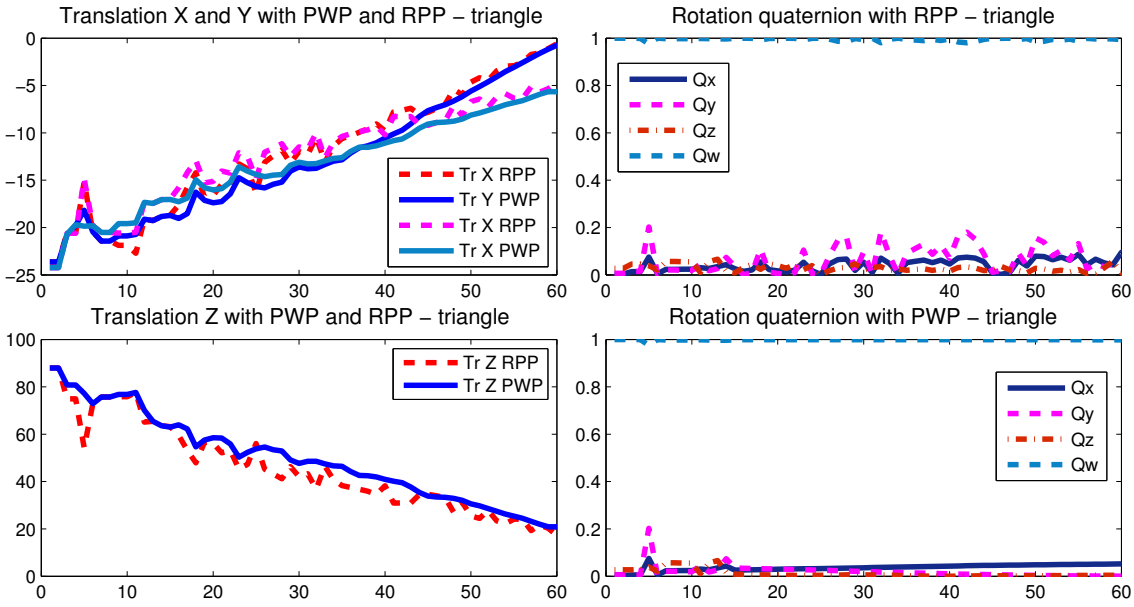


Figure 4.7: System performance while tracking a triangle shaped sign over 60m, with just the 4 point pose recovery (RPP) and with the tracker (PWP).



Figure 4.8: Filmstrip showing frames from a video tracking a pedestrian crossing sign.



Figure 4.9: Filmstrip showing frames from a video tracking a triangular sign.



Figure 4.10: Filmstrip showing frames from a video tracking multiple round signs.



Figure 4.11: Filmstrip showing frames from a video tracking a parking sign, in spite of it being occluded by a tree.

4.3 Human Computer Interaction

A second application of our tracker is in the field of human computer interaction. Here our goal is to replace the standard mouse and keyboard-type interaction, which limits the user to a constrained work flow, with gesture and touch-based interfaces. In this section we describe a natural and intuitive way of interacting with a virtual environment, by combining a vision system based on our PWP3D tracker with an inertial sensor. Our system, while still being cost effective, is able to recover the rigid pose of a human hand, in real time. Furthermore, we do not limit ourselves to a single, fixed, hand configuration. Instead, we are able to distinguish between a small, discrete, subset of such configurations (such as pointing or grasping).

The standard method of doing fast and reliable hand tracking is to use marker or glove-based approaches. One commercial example is the the ShapeHand data glove (Measurand Inc., 2009), which uses a combination of accelerometers, gyroscopes and flex sensors. Two non-commercial examples are Wang and Popović (2009) and Maes (2009). Wang and Popović (2009) use a specially coloured glove (but no sensors) to obtain the full 3D articulated pose of a hand. The SixthSense system of Maes (2009) tracks individual fingers (rather than the whole hand) using specially coloured markers on the tips of the fingers. The system is able to track the markers and recognise several hand gestures in real time and with very high accuracy. Tracking is only 2D i.e. the system provides the position of the fingers in *image space* rather than *full 3D space*.

Vision based 3D hand tracking can be split into *model-based* and *appearance-based* tracking (Pavlovic et al., 1997). A model-based technique has a number of important elements: a 3D model, a set of image features, an error function and a non-linear optimisation algorithm. At every frame the set of features is extracted from the image and an error function is minimised. This energy function relates the pose of the 3D model to the features extracted from the image. The features may be simple (edges, points) or complex (3D depth information). Most often edges and colour are combined to form a silhouette (de Campos and Murray (2006); Lu et al. (2003) and

Lin et al. (2002) among others). The 3D model may be anything between a coarse geometric model and a detailed 3D reconstruction (including shading and texture information) of the user's hand. For example, Stenger et al. (2006) use a model built only from quadratics, while de La Gorce et al. (2008) use both texture and lighting. To form the error function the 3D model may be projected down or the image features may be projected up. The minimisation may lead to one or to several hypotheses. Finally, sometimes the hand motion dynamics are used, therefore predicting the pose of the next frame from the poses at the current frame and previous frames.

Appearance-based algorithms propose to learn a direct mapping between image features and pose. Therefore only two steps are needed when a new frame is presented: extract the image features and obtain the pose from the feature–pose mapping. These methods have the advantage (over model based methods) that no initialisation is need. These approaches are also (theoretically) faster, because most of the processing effort is put into training, which is done offline, rather than in the online processing. To our knowledge there are only a few papers in the 3D hand tracking literature where appearance-based tracking is used. The mapping between feature and pose can either be learnt inside a database (in Athitsos and Sclaroff (2003)), by using a tree-based filter (in Stenger et al. (2006)) or via a Relevance Vector Machine (in de Campos and Murray (2006)). These methods, while maybe producing good results in controlled environments where the segmentation is very good, do not perform well in real-world scenarios.

We use the visual 3D tracker to recover the 6 degree of freedom rigid pose of the hand, with a known (fixed) 3D model and, to help deal with the ambiguities in the pose–silhouette mapping, we place an accelerometer on the hand.

Tracking the full articulated pose from a single, natural, image of the hand remains an open problem. In this work we choose a compromise between rigid and full articulated tracking, by using an SVM one-against-all multiclass classifier to distinguish between a small subset of articulated hand poses and changing the 3D model as a function of the articulated hand configuration. As input to the classifier, we use a histogram of oriented gradients (HOG) descriptor of 2D hand segmentations.

Our method has the following significant advantages over previous hand tracking work:

- unlike most previous work, such as de Campos and Murray (2006); Stenger et al. (2003) and de La Gorce et al. (2008), it works in real time (for fixed hand configurations) and close to real time (for variable hand configurations);
- unlike many edge based works, such as de Campos and Murray (2006), it does not require a prior hand segmentation and is region based, allowing it to work in real-world environments (cluttered and with large amounts of motion blur and occlusions);
- it is much less intrusive, compared to glove-based approaches (such as Measur- and Inc. (2009); Wang and Popović (2009) and Maes (2009));
- it is not limited to a single, fixed articulated hand pose, but rather is able to choose between different 3D models, based on the configuration of the hand, effectively giving the appearance of articulated tracking.

The remainder of the section is structured as follows: we begin in Section 4.3.1 with an overview of our system. In Section 4.3.2 we present the integration of the PWP3D tracker, and in Section 4.3.3 we detail the way in which we extract the orientation from the accelerometer and correlate it with the 3D tracker. In Section 4.3.4 we show how we switch between different 3D models, based on the articulated pose of the hand. Section 4.3.7 shows various results.

4.3.1 Overview

An outline of our algorithm is shown in Figure 4.12. When an image is received from the camera it is piped through a 2D segmentation step, which uses a 2D level set based segmenter to give us an approximate hand segmentation. This is then refined using graph cuts and rescaled to a standard size. A HOG descriptor is built from the normalised segmentation and a multiclass SVM classifier is used to determine the configuration of the hand (out of a fixed, small subset of possible configurations).

The detected configuration determines the 3D model used in the visual 3D tracker, which is then run on the original image from the camera. An accelerometer (placed on the hand) is used to aid the visual 3D tracker.

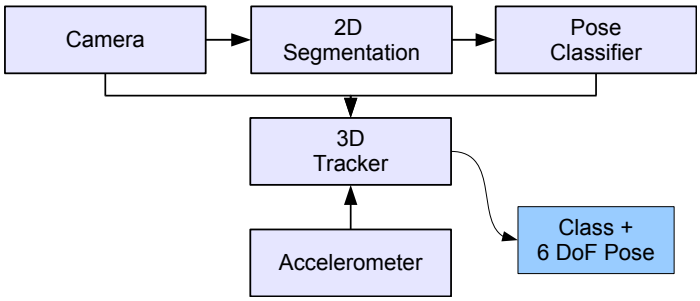


Figure 4.12: System Overview

4.3.2 Visual 3D Tracker

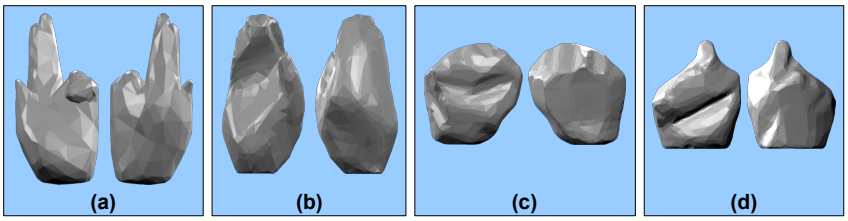


Figure 4.13: 3D Models. (a) is used in Section 4.3.8; (b),(c),(d) are used in Section 4.3.4.

We use our PWP3D tracker and 3D triangle meshes as models, dynamically selecting between various 3D models. Figure 4.13 shows the four models, each with front and back views, that we employed throughout this section. Each model was obtained by taking several pictures of the hand (using a normal digital camera) from different angles and combining them using iModeller 3D (UZR GmbH, 2009), into a coherent 3D model. One source of error in the recovered pose is the difference between the model and the tracked object. Small differences between the real object and the model (in the millimetre range) can lead to errors less than 5-10 degrees for rotation and 5-10 pixels for translation. Naturally large differences, such as not modelling the fingers, leads to complete loss of tracking.

4.3.3 Accelerometer

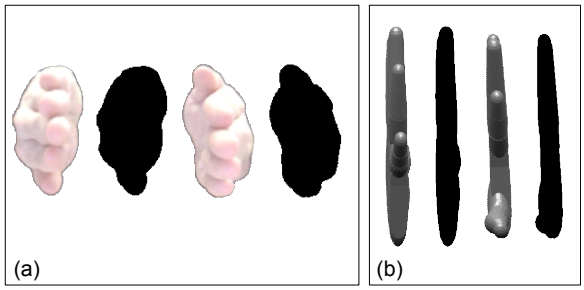


Figure 4.14: Silhouette - Pose ambiguities: (a) real example; (b) artificial example.

Multiple different hand poses can project to the same silhouette, making the mapping from silhouette to pose multimodal. Figure 4.14 shows four such examples, two real and two artificial. The common solution to this problem is to use multiple hypotheses in the minimisation process but a relatively high number of particles is necessary to properly explore the search space. Methods for reducing the number of particles do exist, but even using a relatively small number of eight particles, such as is done by Bray et al. (2004), leads to 8+ times the processing time of our current visual 3D tracker. The solution we employ in this work is to augment the 3D tracker with orientation information from a single off-the-shelf accelerometer, mounted on the hand. The next paragraphs discuss the three aspects regarding our usage of the accelerometer: orientation data extraction, calibration and accelerometer-tracker integration.

Extracting Hand Orientation from the Accelerometer

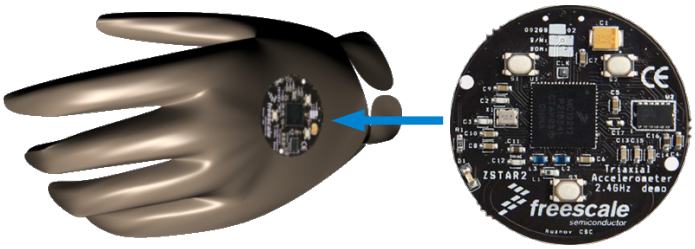


Figure 4.15: Hand with accelerometer sensor board

In this work we use the Freescale ZSTAR3 dev kit. It consists of a sensor board and an USB dongle. The transmission between sensor and dongle is wireless (through the ZigBee / IEEE 802.15.4 (Farahani, 2008) protocol). The sensor board can be mounted either on top of the hand, as shown in Figure 4.15, or in the palm of the hand. Although the wrist would have been a more convenient point of placement (from the point of view of the user), this would measure arm rotation instead of hand rotation.

This accelerometer measures the direction of gravity summed with the instantaneous linear acceleration. Thus, in the steady state (zero or constant velocity) it returns the "down" (world z) direction.

The absolute orientation on two axes (pitch and roll) can be computed from the gravity vector using the standard Euler angle formulas (Freescale, 2007). Note, however, that rotation around the z axis, in world coordinates (yaw) is not measurable. To avoid issues that arise with angle wrap-around and singularities in the Euler angle representation, we use quaternions and convert the current and previous gravity vector measurements into a relative angle change. Also note that this is an approximation, since we are in effect ignoring any linear acceleration the hand undergoes.

Accelerometer - Tracker Calibration

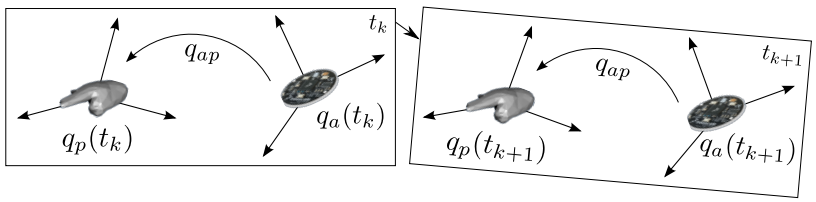


Figure 4.16: Rotation quaternions and coordinate systems for the visual 3D tracker and the accelerometer

A fundamental problem when combining data from inertial and visual sensors is calibration. Each sensor outputs a value for rotation, under some representation, in its own coordinate system. In order to combine the two, the values need to be converted to the same coordinate system i.e. the (constant) offset between the various rotation

sources needs to be obtained. In our case, this is the quaternion offset q_{ap} between the quaternion q_a given by the accelerometer and the quaternion q_p given by the visual 3D tracker (as shown in Figure 4.16). For a time step t_k, we can therefore write:

$$q_a(t_k)q_{ap}(t_k) = q_p(t_k) \tag{4.6}$$

To collect q_{ap} values, we measure q_a and q_p as the user moves his/her hand trough the entire range of orientations. Figure 4.17 shows an example such motion.



Figure 4.17: Example calibration motion

A standard way to solve Equation 4.6 for a set of q_{ap} values is to use the (linear or nonlinear) least squares framework (Hol et al., 2010; Lang and Pinz, 2005). For example, Lang and Pinz (2005) maximise:

$$q_{pa} = \arg\max_{q}\{q^T\Big(\sum_{i=1}^{t}\overline{Q}^T_{\Delta q_a}(t)Q_{\Delta q_p}(t)\Big)q\} \tag{4.7}$$

where $q_{pa} = q_{ap}^{-1}$, Q and $\overline{Q}$ are the matrix representations of Δq_a and Δq_p, and:

$$\Delta q_p = q_p^{-1}(t_1)q_p(t_2) \quad \Delta q_a = q_a^{-1}(t_1)q_a(t_2) \tag{4.8}$$

Our solution assumes these values lie close together on the rotation sphere and we choose the minimum principal component of the q_{ap} data set as our calibration quaternion. The first three principal components will capture the variance inside the data set (hopefully small), while the forth (and minimum) principal component will effectively be a noise-corrected mean quaternion.

Denoting by $\mathbf{Q}$ the $t \times 4$ matrix composed by "stacking" together t quaternions,

our method can be detailed as:

- Subtract the mean quaternion from each row in the matrix $\mathbf{Q}$;
- Compute covariance matrix for $\mathbf{Q}$: $\mathbf{C} = \frac{1}{t}\mathbf{Q}^T\mathbf{Q}$;
- Compute the eigenvectors $\mathbf{V}$ and eigenvalues λ for the matrix $\mathbf{C}$;
- Choose the eigenvector with the minimum eigenvalue as the calibration quaternion.

Figure 4.18 shows a comparison between our calibration method and the least squares one of Lang and Pinz (2005). We use two rotations, over 500 frames, to generate two sets of 500 quaternions, $q_1(t)$ and $q_2(t)$. The calibration quaternion between the two rotations is constant and known. We apply uniformly distributed random noise, at each frame and on each axis, within the [-5°, 5°] range. Next we recover q_c, where $q_2(t) = q_1(t)q_c$, i.e. the calibration quaternion, with both methods. At this noise level, both methods yield similar results. We then add a secondary source of noise, a uniformly distributed random noise, at every 60th frame and on each axis, within the [-30°, 30°] range. At this noise level, our method still successfully obtains q_c, while the method of Lang and Pinz (2005) fails.

Ideally the user would calibrate the system before every use, where, typically, the calibration process would take around 30-40 seconds. In our experiments we have noted however that a single, one time, calibration is enough. As long as the user places the accelerometer in roughly the same position and orientation on the hand (within 1 centimetre and 5 degrees, on each axis, from the calibrated pose), the visual tracker is able to compensate for any error in angle. This happens because we only use the accelerometer as a starting point for the visual tracker optimisation, which we discuss in the next subsection.

Accelerometer – Tracker Integration

To smooth the accelerometer output, we use a Kalman Filter, with a constant acceleration motion model on the raw accelerometer data. Even so, the actual output of

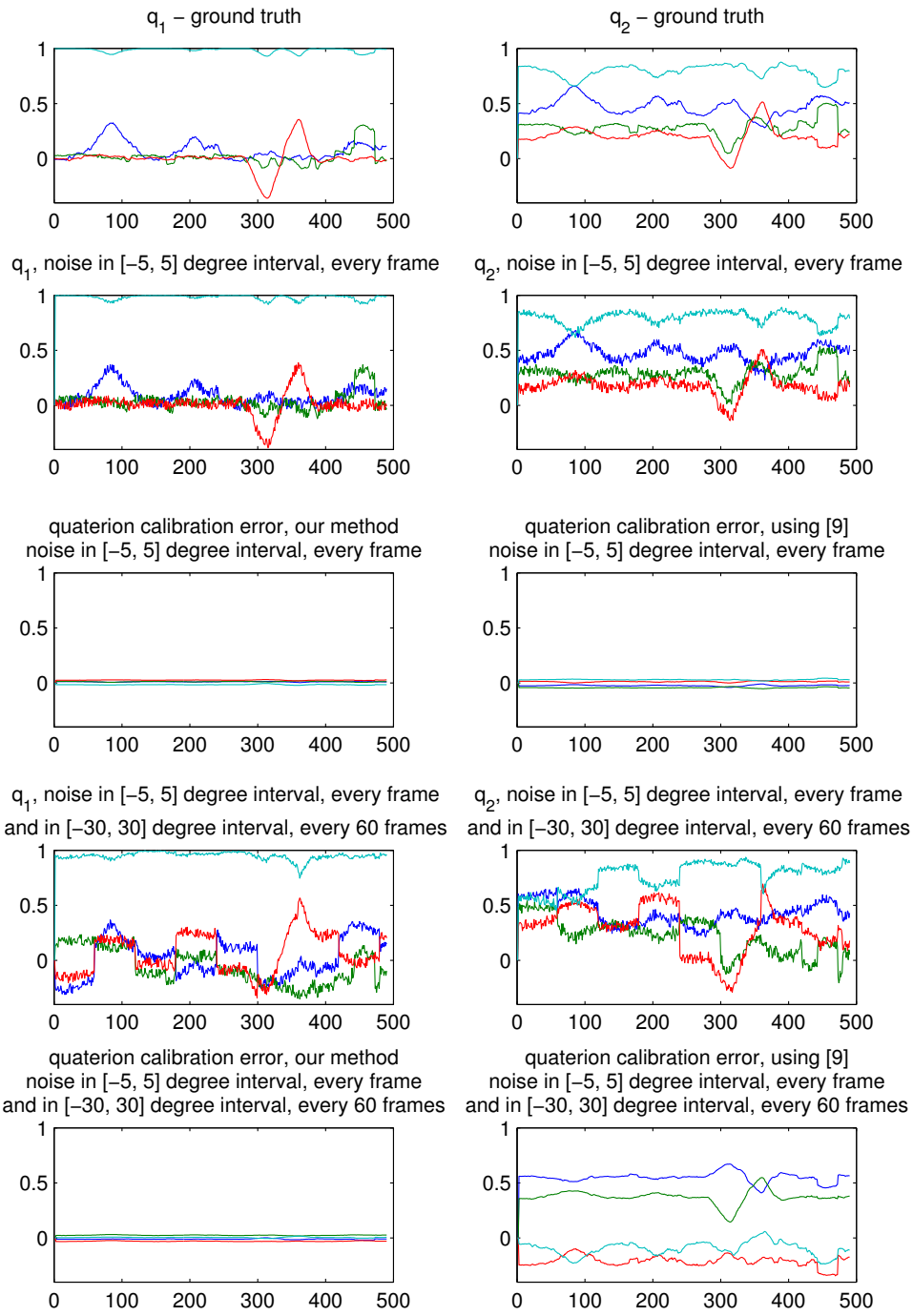


Figure 4.18: Charts comparing our calibration method to Lang and Pinz (2005). We use two known 500 frame motions, q_1 and q_2, with a known calibration quaternion. We first add a small amount of noise at each frame (random angle in [-5°, 5°]) and compute the calibration quaternion, using the two methods. Both recover the calibration quaternion successfully. Next, we keep this source of noise and add another, bigger, one every 60 frames (random angle in [-30°, 30°]). Our method again recovers the correct calibration quaternion, while the method of Lang and Pinz (2005) fails.

the accelerometer is not especially reliable, for three reasons: (i) it remains noisy; (ii) it cannot measure one of the three axis of rotation and (iii) it measures the gravity vector alone only in the steady state, but, when subjected to acceleration, it measures the sum of this acceleration and the gravity vector. Therefore we trust the accelerometer data only as a starting point for the visual 3D tracker iterations. We begin with the rotation estimate from the previous frame, update it with the differential motion obtained from the accelerometer, and use this as the starting point for the visual 3D tracker's iterations. This increases the speed and reliability of convergence of the visual 3D tracker as well as overcoming many of the visually ambiguous poses, but places little faith on the actual output of the accelerometer. Note that we used a similar method of integration in the traffic sign tracking application. There the detector aided primarily in obtaining the correct translation (by relocalising the 3D tracker), while here the accelerometer helps by providing a rotation relocalisation.

4.3.4 Pose Classifier

The full articulated pose of the hand has 27 degrees of freedom, split between the 6 necessary to describe the pose of the palm (i.e. the rigid pose) and the 21 needed to describe the articulation of the phalanges. There are several problems when dealing with articulated hands and the high dimensional spaces they live in: (i) local minima: searching in a 27 dimensional space is considerably more likely to run into local minima than searching in a 6 dimensional space; (ii) multimodality: as explained above, multiple poses can generate the same silhouette and (iii) speed: tracking speed is inverse proportional to the number of objects tracked i.e. tracking all 19 bones in the hand will be much slower than tracking a single object.

Often however, the number of possible hand configurations is limited by the activity the hand is engaging in. For example, when picking up a cup, the hand pose will be limited to a small, discrete subset, rather than the full, high dimensional, space. Based on this idea, in this work, we propose a mixture between appearance based and model based methods, which solves some of the above mentioned problems: (i) we segment the hand in 2D and use a multiclass classifier, combined with local features

within the boundary of the segmentation, to infer one of several possible hand configuration and (ii) we use separate 3D models for each configuration and the visual 3D tracker (and accelerometer) to track the rigid hand pose parameters. Compared to the full 27 dimensional hand tracking, our solution is a compromise, but is far less likely to run into local minima and can operate at close to real time speeds. A similar methodology, though using a hierarchy of models and a different classifier and not coupling with a 3D rigid pose tracker, was used by Stenger et al. (2006).

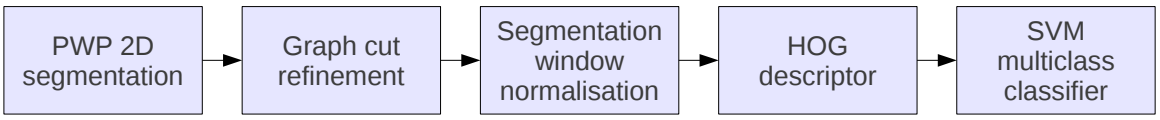


Figure 4.19: Pose classification overview

The remainder of this section details each step of our pose classifier, as depicted in Figure 4.19.

2D Hand Segmentation

The first step of our proposed solution is to segment the hand. For this we use the segmentation phase of the PWP tracker of Bibby and Reid (2008) (upon which our PWP3D tracker is based) i.e. an unconstrained evolution of a pixel-wise posteriors level set energy function:

$$E(\Phi) = -\sum_{\mathbf{x}\in\Omega}\left(\log\left(H_e(\Phi)P_f + \big(1 - H_e(\Phi)P_b\big)\right) - \frac{(|\nabla\Phi(\mathbf{x})| - 1)^2}{2\sigma^2}\right) \tag{4.9}$$

The difference between this energy function and the PWP3D one is the added geometric prior on Φ, which rewards a signed distance function. All other terms have the same meaning (see Chapter 3).

The unconstrained evolution is done by differentiating Equation 4.9 w.r.t. the level set function:

$$\frac{\partial P(\Phi|\Omega)}{\partial\Phi} = \frac{\delta_e(\Phi)(P_f - P_b)}{H_e(\Phi)P_f + \big(1 - H_e(\Phi)\big)P_b} - \frac{1}{\sigma^2}\left[\nabla^2\Phi - \mathrm{div}\left(\frac{\nabla\Phi}{|\nabla\Phi|}\right)\right] \tag{4.10}$$

For more details, the reader is referred to Bibby and Reid (2008).

The PWP algorithm combines a tracking and a segmentation phase. The tracking phase is very fast (running at over 100Hz), but the segmentation phase can be slow as it can require many iterations for full convergence. Since we use the 2D tracker to provide segmentation, we would need to iterate the level set minimisation to complete convergence, making the full PWP segmentation too slow for our needs. Our solution is to obtain coarse segmentations (Figure 4.20) using the PWP tracker, by running a fixed, small, number of iterations for the level set evolution, and refine the segmentation using graph cuts.

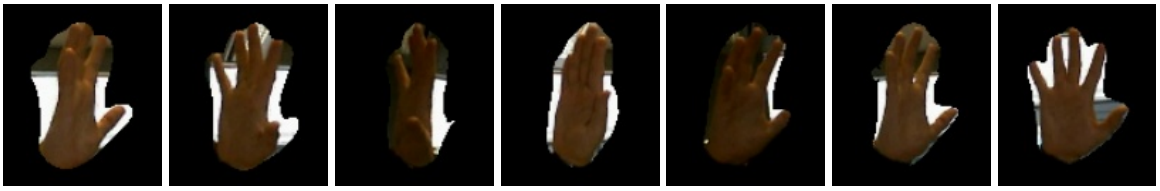


Figure 4.20: Example coarse hand segmentations obtained by the PWP algorithm.

Using to the output from the level set tracker and Equation 3.5, we define a per pixel trimap. Foreground pixels (those that have $P_f >> P_b$) are connected to the source with a very high, preset weight and to the sink with weight 0, background pixels (those with $P_b >> P_f$) the opposite. The remaining pixels are connected to the source with weight P_f and to the sink with weight P_b. The binary potentials between neighbouring pixels is the L2 distance between their RGB colour values. We then use the Boykov and Kolmogorov (2004) Min-Cut/Max-Flow algorithm to obtain the final segmentation.

The final segmentation windows are normalised to a fixed size (112×128 pixels, in this work). Examples of final, normalised segmentations are shown in Figure 4.21.



Figure 4.21: Example final segmentation results.

4.3.5 Descriptor

The next step is to build histogram of oriented gradients (HOG) (Dalal and Triggs, 2005) descriptors for the normalised segmentations.

HOG divides each descriptor window into groups of pixels, called cells. For each cell, a histogram of gradient orientations is computed. We use 4×4 pixel cells and 5 histogram bins, evenly spaced between 0 and 180 degrees. Multiple cells are grouped into blocks, with each cell being part of multiple blocks. We use 2×2 blocks. Finally, each block histogram (consisting of 2×2 cell histograms) is normalised using the L2-Hys norm (Dalal and Triggs, 2005; Lowe, 2004).

4.3.6 Multiclass Classifier

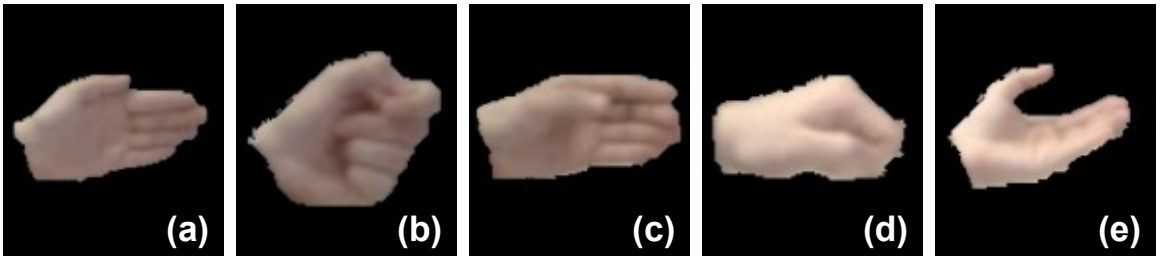


Figure 4.22: Example training data: (a) – class 1; (b) – class 2; (c) – class 3; (d), (e) – negative class. The models for (a), (b) and (c) are shown in Figure 4.13.

The final step in our algorithm is to use the one-against-all multiclass SVM algorithm of Crammer and Singer (2001). In this example we use 4 classes (Figure 4.22): three for the various hand configurations (Figure 4.13 (b), (c) and (d)) and one for unmodelled configurations. The classes are trained to recognise a fixed hand shape in a specific pose. Based on the output from the classifier, we change the 3D model for the visual 3D tracker: when one of the positive classes is detected we change accordingly, when the negative class is detected (the hand with an unfamiliar shape and/or in an unfamiliar pose) we leave the 3D model unchanged.

For both the descriptor computation and the classification, we developed our own GPU based implementation. For more details on the implementation see Appendix A.1. The processing time for both stages (summed together) is usually less than *2ms*.

4.3.7 Results

In this section we show the results obtained by applying our algorithm to various video sequences and hand configurations. We showcase the benefits of using a region based method and the accelerometer, and then show a typical failure case. This is followed by an example of our algorithm successfully tracking different articulated hand poses. Next we show a quantitative comparison between the poses produced by our tracker (with and without the accelerometer) and the ground truth. We finish by detailing a proof-of-concept user interface, based on our 3D tracker.

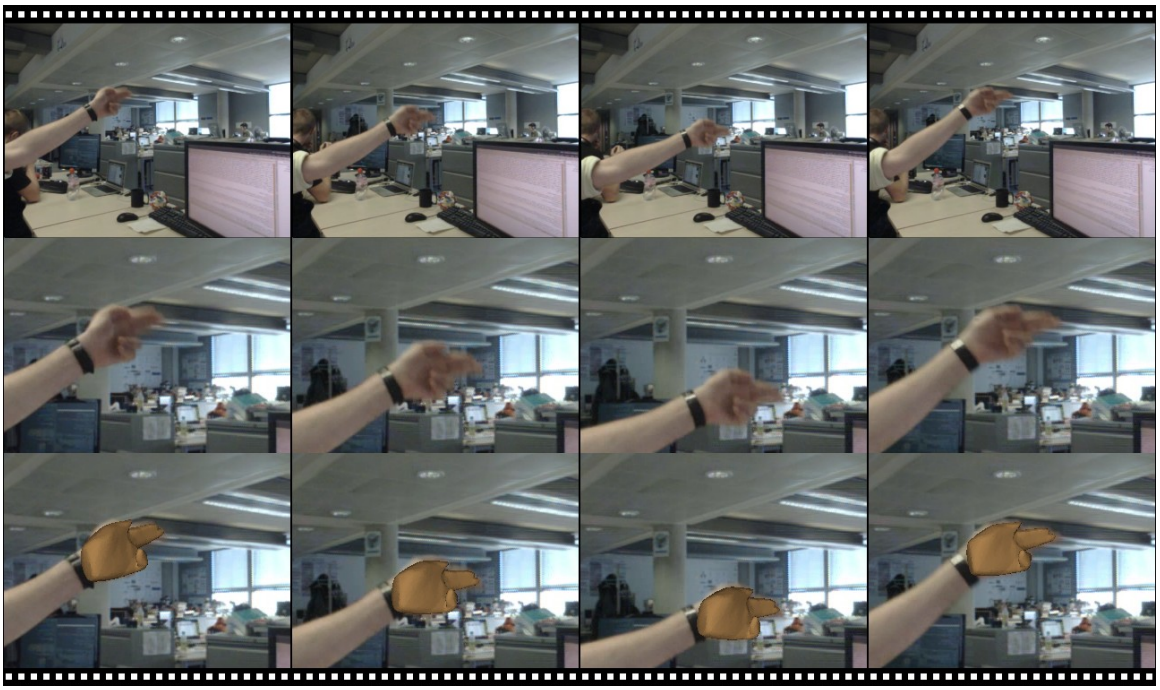


Figure 4.23: Filmstrip showcasing robustness to motion blur. The top row shows the original camera image, the middle row shows a zoomed snapshot of the hand and the bottom row shows the result obtained by our algorithm.

A region based tracker has several advantages: robustness to motion blur, occlusions and cluttered environment. Experiments showcasing the first two are shown in Figures 4.23 (motion blur) and 4.24 (occlusions). Cluttered environments are present throughout the results section, in Figures 4.23, 4.24, 4.25 and 4.28.

Figure 4.25 and Figure 4.26 contain two filmstrips, showcasing the benefits of using the accelerometer. In Figure 4.25, for each frame, we show the original camera

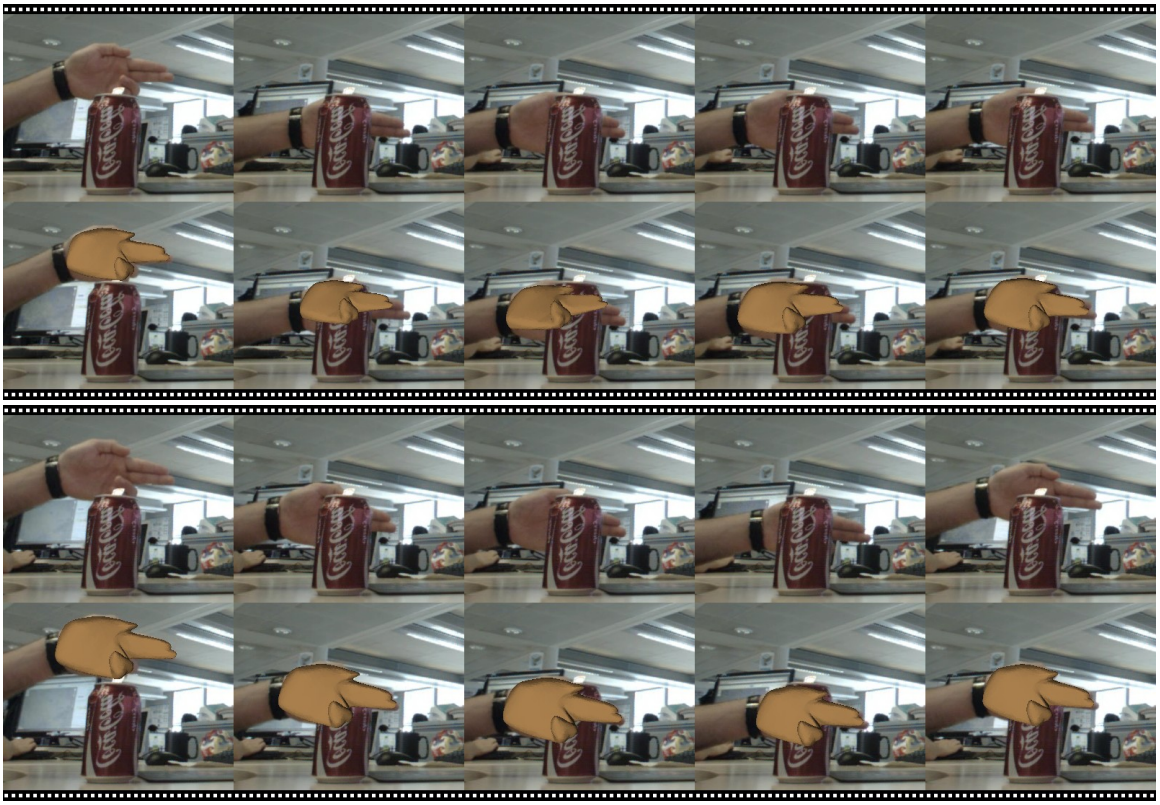


Figure 4.24: Robustness to occlusions: top filmstrip – as the hand moves behind the soft drink can, the colour models have not yet adapted and the results are incorrect; bottom filmstrip – after the histogram has adapted the results correct.

image and two OpenGL renderings of the hand, with the determined pose, with and without the accelerometer. In Figure 4.26, for each frame, we show a posterior map (the per pixel difference $P_f - P_b$, where $P_f - P_b > 0$, with P_f and P_b defined according to Equation 3.5) and two wireframe renderings, created by our rendering engine and used by algorithm, also with the determined pose and with and without the accelerometer. The posterior map is a measure of how effective the regions statistics are at separating foreground from background. In the first example the hand was first moving up/down and then left/right, very quickly, in a frontal camera view. In the second example the hand was moving up/down, but at a side camera view. Each motion took up to 5 frames at 30 fps video (so 1/6 seconds), followed by a short (around 60 frames) period of stability. In both examples, initially, when there are no ambiguities, the results are similar, with and without the accelerometer. At the

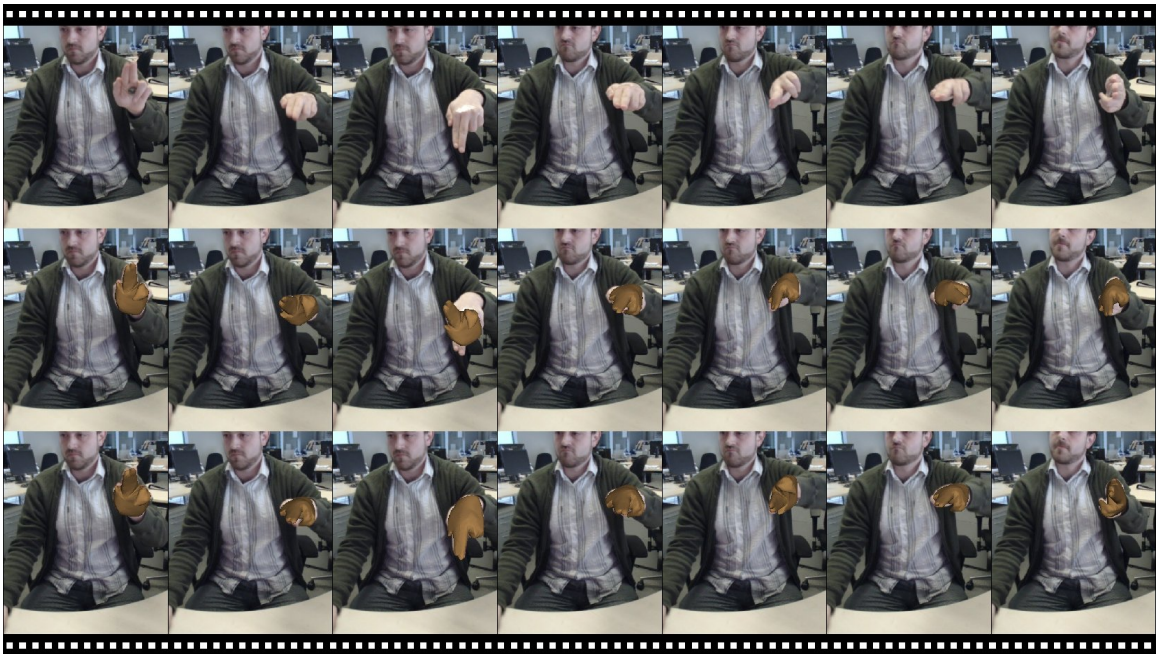


Figure 4.25: Filmstrip showing tracking results with and without the accelerometer. The top row shows the original camera image. The middle and bottom rows show OpenGL renderings of the hand in the recovered pose, without and with the use of the accelerometer, respectively. The visual 3D tracker alone cannot deal with ambiguities in the silhouette (where a change in pose does not alter the silhouette). The accelerometer provides enough information to discriminate between the ambiguous poses.

second frame, also in both examples, it can already be seen that the visual tracker alone is showing some error in the recovered pose. This can be explained in both cases by the fact that, when the accelerometer is not used, the visual tracker does not have time to converge fully, whereas, when the accelerometer was used, the starting point is closer to the actual pose, so fewer iterations are required. At the next frame, again in both examples, the pose recovered by the visual tracker is very far from the correct one. In the first example, the frontal view of the hand is an ambiguous pose i.e. a change in hand rotation will not alter the silhouette. In the second example, the visual tracker converges to a local minimum because it was not allowed to converge fully at the previous frame. In the first example the hand moves back up, allowing the visual tracker to recover, albeit not perfectly. Again, when the accelerometer is used, the recovered pose is more accurate. The same behaviour can be noticed in the left/right rotation (next 4 frames of the first example). In the second example the

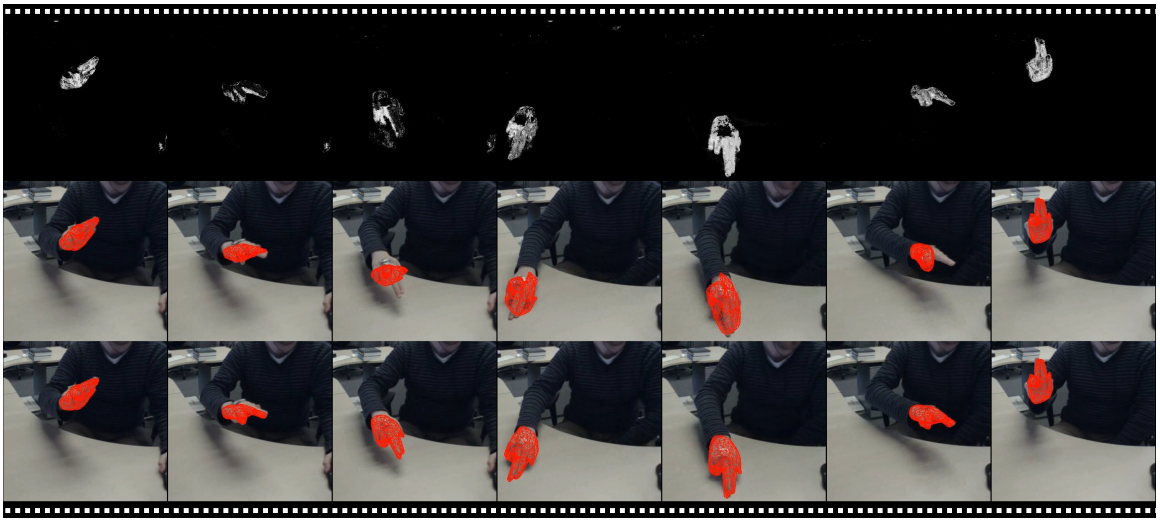


Figure 4.26: Filmstrip showing tracking results with and without the accelerometer. The top row shows the posterior map (the per pixel difference $P_f - P_b$, where $P_f - P_b > 0$). The middle and bottom rows show wireframe renderings made by our rendering engine and used by the visual tracker. The maximum allowed number of iterations was not enough for the visual tracker alone to converge, which quickly finds an incorrect local minima. The accelerometer positions the visual tracker closer to the correct result, so fewer iterations are required and the results are better.

hand does not go back to a pose where the visual tracker can recover, so the results are incorrect for a longer period of time.

Next we show a typical failure case, in Figure 4.27. The accelerometer only provides information on two axis, pitch and roll. Its measurements are not affected by changes in yaw. This means that, when the visual tracker encounters ambiguities on the yaw axis, it will no be able to disambiguate between them. The algorithm can also fail in recovering translation, when the hand moves fast enough for the projection not to overlap with the object in the image, at consecutive frames.

In Figure 4.28 we show our system correctly classifying between different configurations (articulated poses) of the hand, and switching to the correct 3D model. The top row shows the camera image together with the normalised descriptor (bottom left). The contour of the projection is also overlayed, in red. The middle row shows the recovered pose, with the appropriate 3D model. In this example we used four classes, three positive and one negative. When the classifier finds a negative class

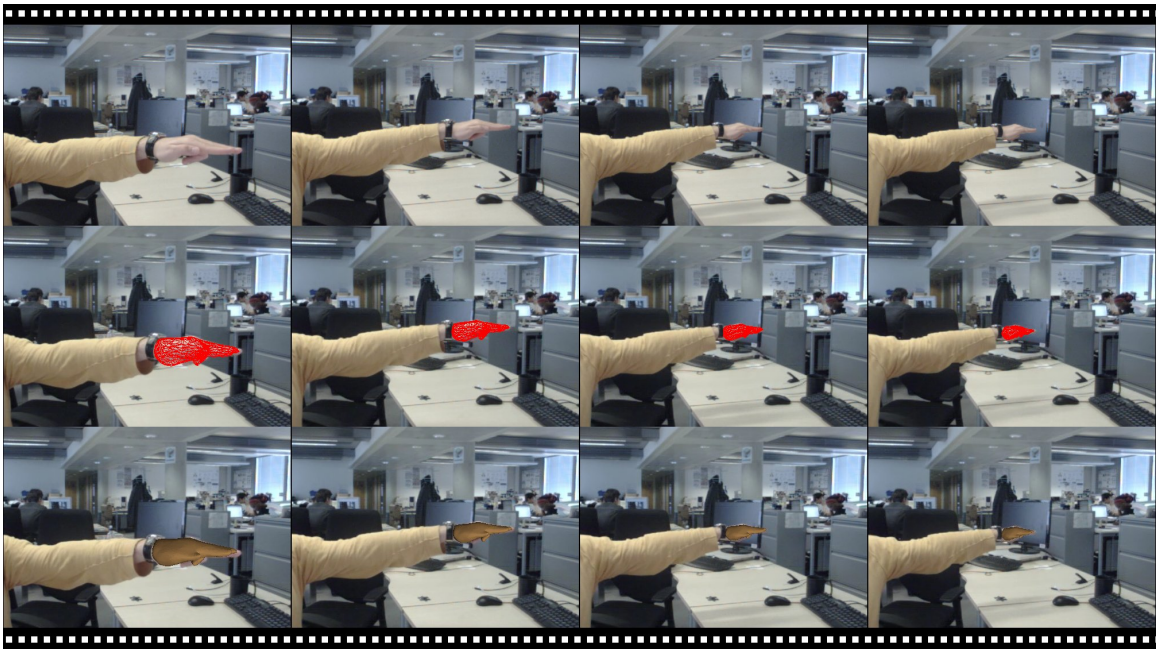


Figure 4.27: Filmstrip showing tracking a typical failure case. The accelerometer does not provide any yaw, so it cannot help the visual tracker disambiguate between similar silhouettes.

frame, it uses the class and 3D model from the most recent frame. The bottom rows show a proof-of-concept augmented reality application, where such a technology could be used, by rendering a virtual cup and a virtual tennis racket, as it would appear if it were held by the hand.

4.3.8 Quantitative Benchmark

Quantitative comparison against absolute ground truth is particularly challenging, and to our knowledge, no datasets exist that would enable a fair comparison. To that end, here we present a methodology for testing our system against an alternative input method. We generate absolute positions of a fingertip by moving a hand on the surface of a tablet pc, simultaneously tracking the hand using our system (both with and without the accelerometer). Using the robust planar pose algorithm of Schweighofer and Pinz (2006), we recover the pose of the tablet pc touch screen relative to the camera, which allows us to compare two sets of poses in the same coordinate system. For this experiment we used an HP TouchPad tablet pc, running



Figure 4.28: Our system tracking multiple hand configurations and proof-of-concept augmented reality application.

Android 2.3.7. Strictly speaking, the touch pad does not provide an absolute ground truth and its accuracy is limited by the area of the finger in contact and the resolution of the touch-sensitive array on the screen. To measure the uncertainty, the user was instructed to tap the screen at various points and we measured the standard deviation of the user's input accuracy. This then yields a confidence interval for the "ground truth" tablet measurements against which we compare our method. This is in effect the area of the tip of the finger, being equal to 5.0 millimetres (x axis) and 4.2 (y

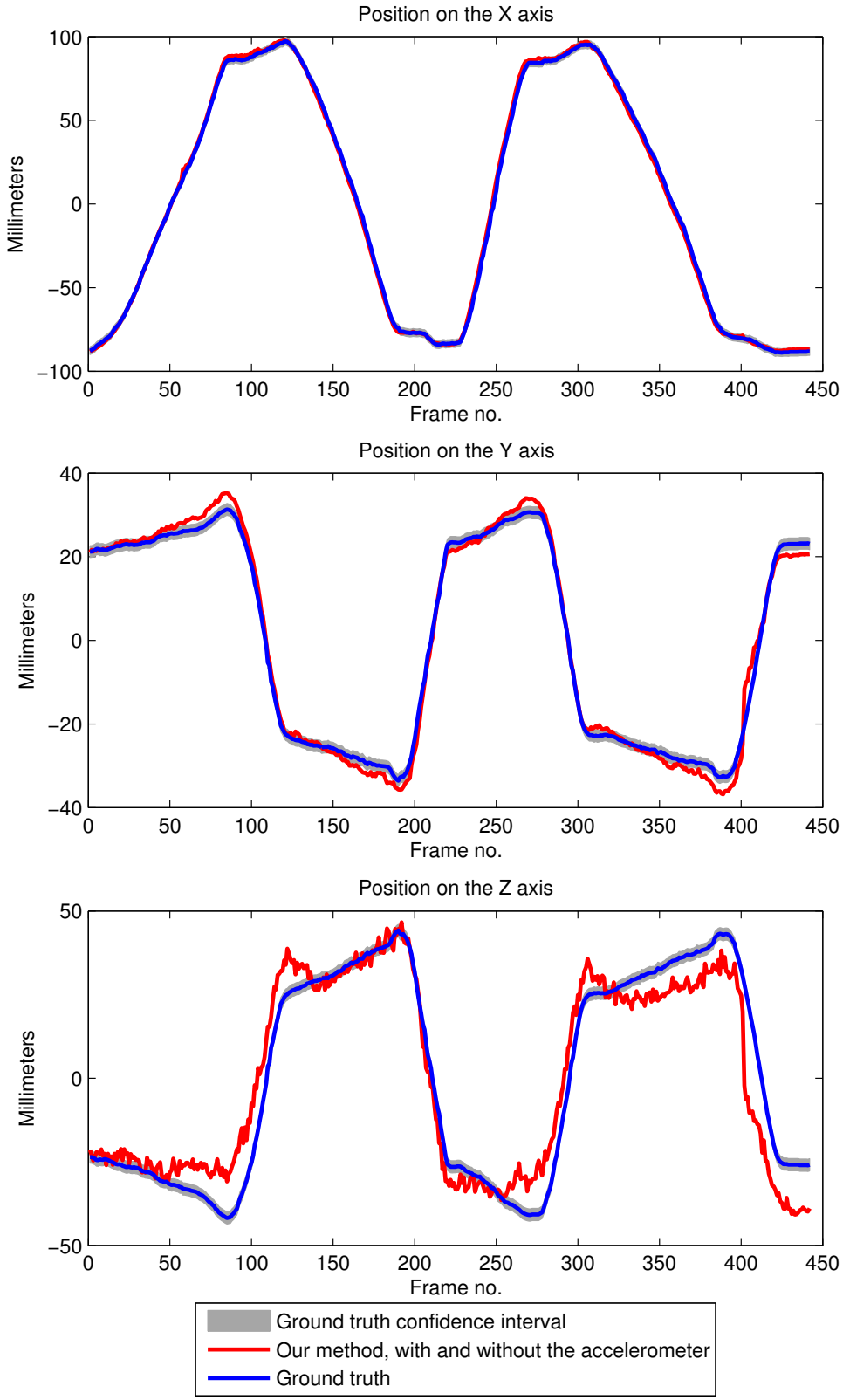


Figure 4.29: Comparison between our method and the ground truth, focusing on translation.

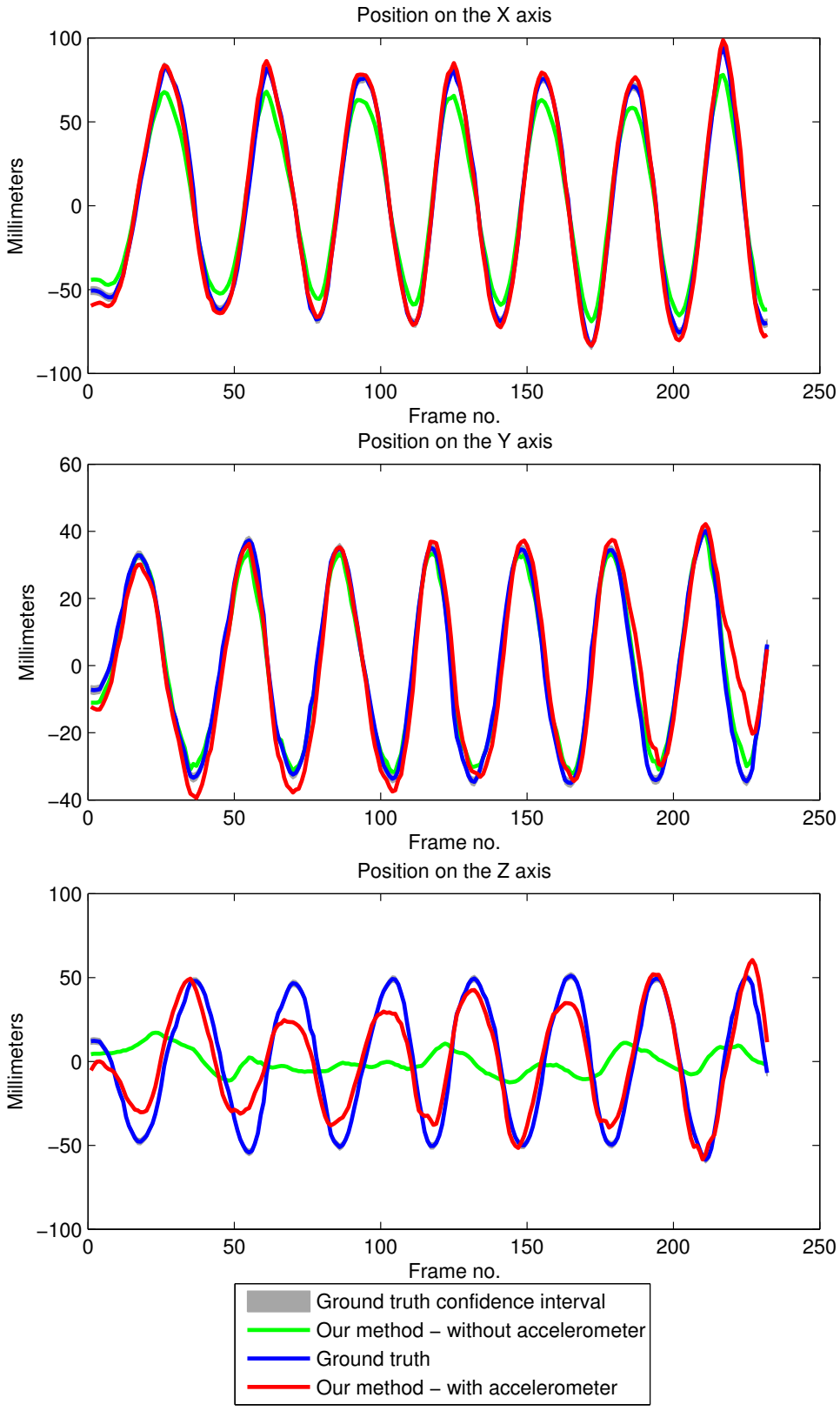


Figure 4.30: Comparison between our method and the ground truth, focusing on rotation.

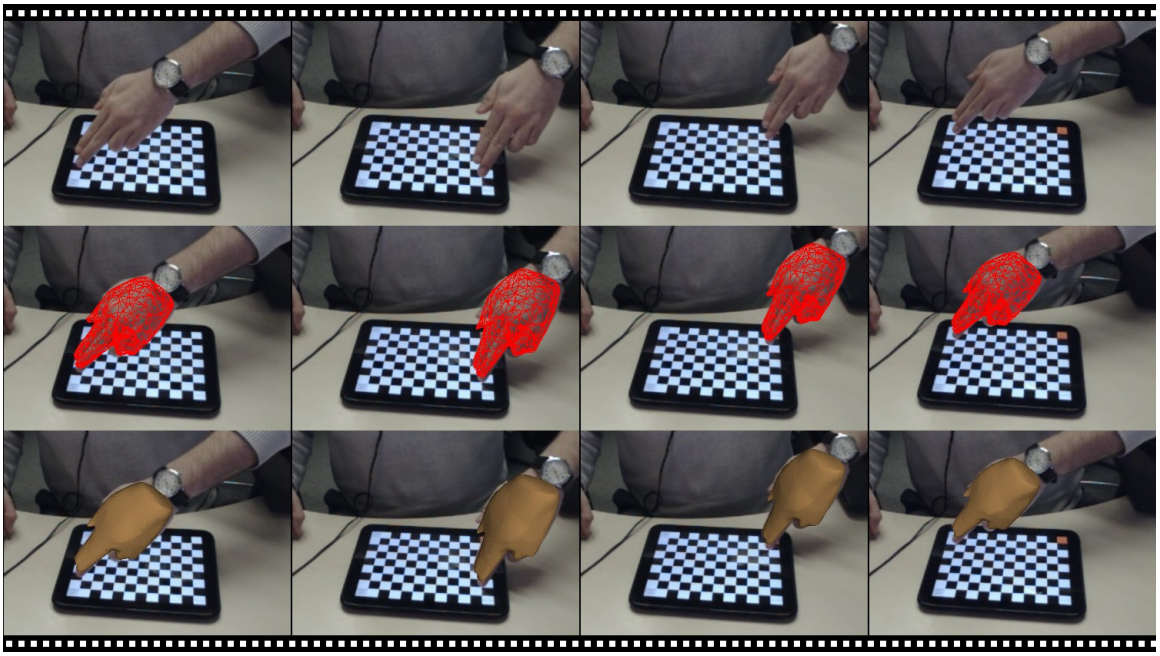


Figure 4.31: Example frames and results from the translation accuracy test. Original image (first row), wireframe (second row) and full rendering (third row) results for our method, with and without the accelerometer.

axis) in touch screen 2D coordinates or 4.6 (x axis), 2.8 (y axis) and 3.6 (z axis) in 3D camera coordinates.

Our first experiment focused on translation. Figure 4.29 shows quantitative results and Figure 4.31 shows example frames and qualitative results. Here the hand was moved in a rectangular pattern, aiming to keep it at a fixed angle. Since we only use the accelerometer readings to provide between frame rotation data, they were not necessary for this experiment. We obtain errors of 1.6 millimetres (x axis), 1.8 millimetres (y axis) and 7.2 millimetres (z axis).

The second experiment focused on rotation. Figure 4.30 shows the quantitative results and Figure 4.32 shows example frames and the qualitative results. Here the hand was moved in a circular pattern, aiming to vary only the angle. Without the accelerometer, the method undergoes catastrophic failure, the mean error on the z axis being 31.8 millimetres. While the errors on the other axis are much smaller (8.5 millimetres (x axis) and 2.2 millimetres (y axis)), the recovered pose would not be usable for any human computer interaction system. Failure here is caused by

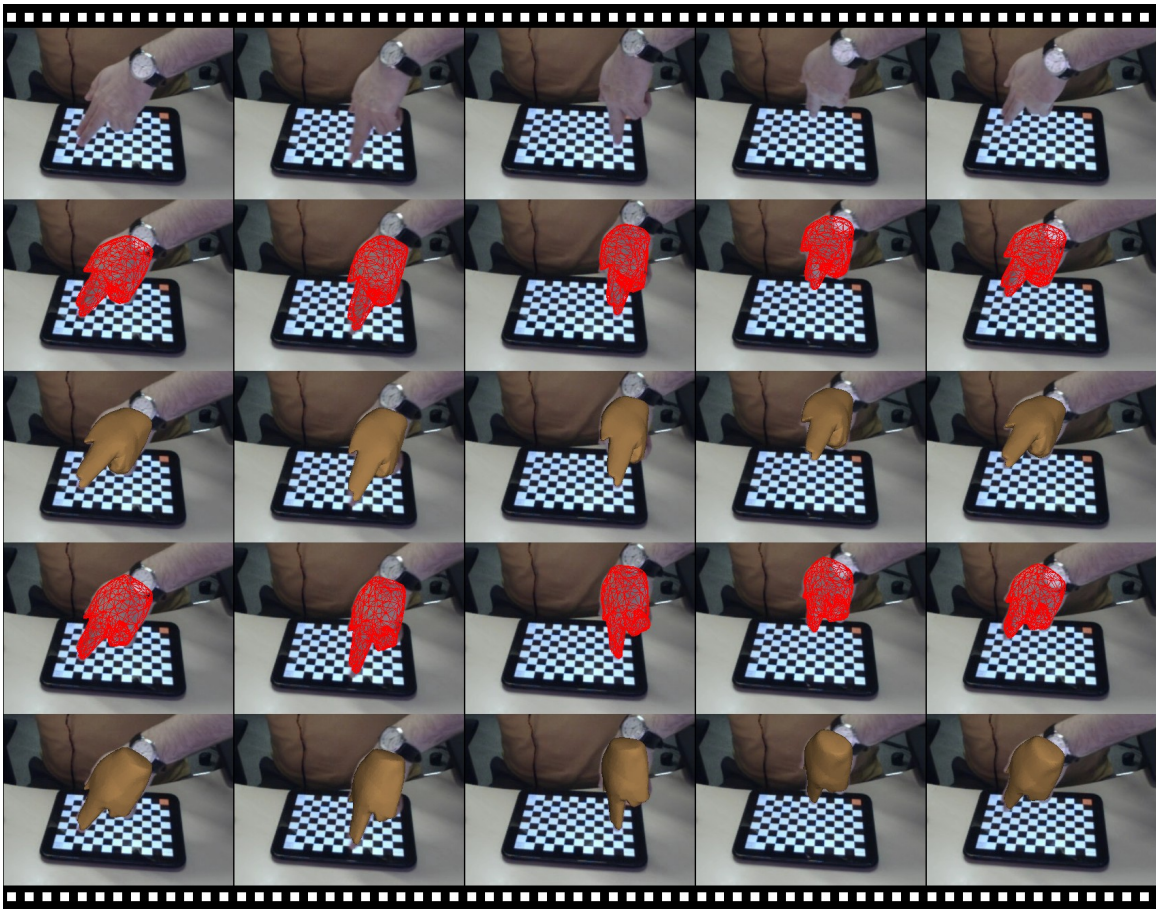


Figure 4.32: Example frames and results from the rotation accuracy test. Original image (first row), wireframe (second row) and full rendering (third row) results of our method, without the accelerometer, wireframe (forth row) and full rendering (fifth row) results of our method, with the accelerometer.

ambiguities in the silhouette to pose mapping, as, on the one hand, there exists an ambiguity between scale (z axis) and rotation and, on the other hand, between different values for rotation. When using the accelerometer the error on the z axis decreases considerably, to 10.6 millimetres. The errors on the x and y axis remain comparably small: 3.1 (x axis) and 5.0 (y axis).

Proof-of-concept Human Computer Interface

In Figure 4.33 we show several filmstrips depicting a proof-of-concept human computer interface based on our hand tracker. The user first has to mark a plane, by moving the hand along the edges of a rectangular board (Figure 4.33 – top filmstrip).

The shape and colour of the board do not influence the hand tracking in any way. The plane does not need to lie on a real surface i.e. it could have been marked in mid-air. We chose this shape as a visual cue to the user and because it provides an easy and structured way of displaying information. The calibration motion could have been different (i.e. a circle). To extract the plane position we used RANSAC on the 3D positions of the tip of the index finger (which we computed using the pose of the full hand and the 3D model). After the plane has been marked, it can be "clicked", by moving the hand so that the distance between the tip of the index finger and the plane is 0 or negative. We display a menu interface on the virtual surface, which the user navigates by clicking. The UI is displayed on the computer screen but could also be projected on the board or shown through a pair of AR glasses.

In the first example (Figure 4.33 – middle filmstrip) the user is prompted to place a DVD on the marked surface. We use two video game DVDs as examples. The system identifies the DVD and displays relevant information on the virtual surface. Recognition is achieved by using the TinEye API (TinEye, 2010) and the information is extracted from sources such as Amazon and Wikipedia. The user is also shown a 3D object, related to the identified DVD. For the first DVD, the main starship from the game is shown and for the second DVD a 3D model of planet Mars. The user can interact with the 3D object. In the case of the first DVD the user can place the hand under the starship which then "climbs" on top of the hand. The user can move the ship by moving the hand. In the case of the second DVD the user can rotate the model of planet Mars by moving the hand.

In the second example (Figure 4.33 – bottom filmstrip) the user first draws a 2D closed shape on the virtual surface. The 2D shape can be turned into a 3D object (by clicking inside it and raising the hand above the virtual surface). The 3D object can then be picked up, similar to the previous example (by placing the hand under it and waiting for it to "climb" on top of the hand).

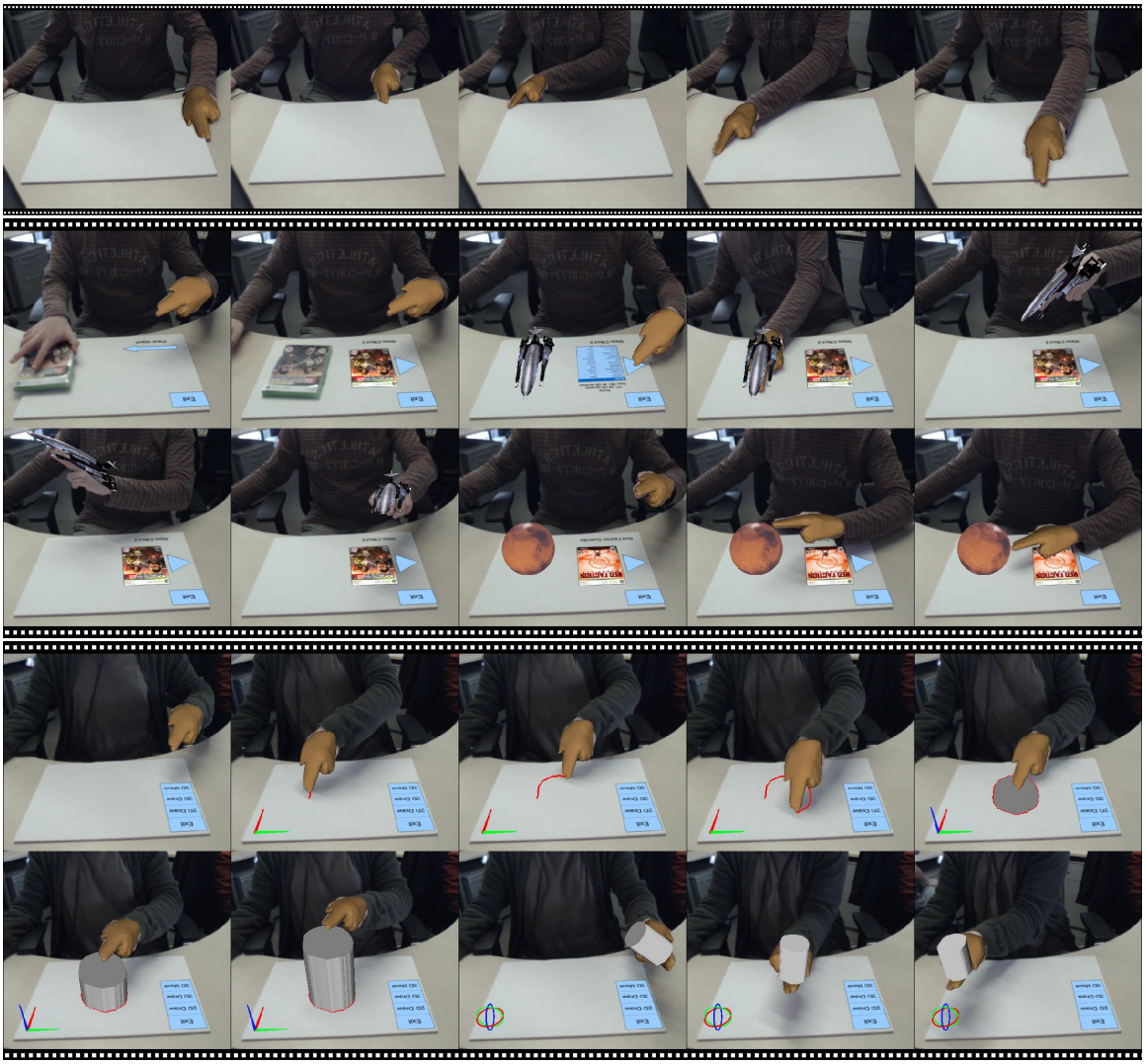


Figure 4.33: Example human-computer interface based on our tracker. Top filmstrip – calibration – the user marks a plane by moving the hand along the edges of the board. Middle filmstrip – the user places a DVD box on the surface, prompting the system to show facts about the DVD box and a 3D object related to it, which can be manipulated with the hand. For the first DVD, the user can interact with a 3D model of a starship. For the second DVD, the user can interact with a model of planet Mars. Bottom filmstrip – the user can draw on the 2D surface, transform the 2D drawing into a 3D object (by tapping inside it and lifting the hand) and pick up and move the 3D object.

4.4 Conclusions

In this chapter we presented two example applications of our PWP3D tracker.

The first one, a driver assistance system, links our tracker to a single view de-

tector. This helps avoid loss of tracking due to lack of overlap in the position of the tracked object at consecutive frames. The second application, a 3D rigid hand tracker, augments the tracker with data coming from an off-the-shelf accelerometer, placed on the hand. This helps us to partially deal with ambiguous poses. We also show how limited, activity specific, faux articulation support could be added to our tracker, by using a multiclass classifier to select between a number of discrete hand configuration.

There remain a number of significant failure cases. The first of these is that we rely on overlap between the predicted and the actual position of the object in the image. This is a problem when the motion is very fast, or when the object is small in the image. This latter case is regularly manifested in the driver assistance system, for distant signs, especially during car rotation. A second source of failure is the multimodality in the silhouette to pose mapping. The accelerometer can often help deal with this but it cannot measure yaw, so the 3D tracker can still fail to reach a correct pose. One possible solution would be the use of a more complex inertial measurement unit (such as a gyroscope). A third source of failure stems from the fact that our our pose classifier is limited to a discrete hand configuration space. This works well for a variety of activities, since people often move their hands rapidly between "standard" shapes, such as open hand and grasping, and our method could be easily extended to more configurations if required. However if continuous articulations of a fully dexterous motion must be tracked, modelling the continuous changes in some manner may be a better long-term approach.

In the next chapter we propose a solution for modelling continuous articulations and deformations, by using dimensionality reduction to learn latent shape spaces.

Chapter 5

Segmentation and Tracking with Shared Shape Spaces

In this chapter we propose a method for monocular tracking of 2D or 3D deformable objects. We capture intra-activity or intra-class shape variation using low dimensional latent spaces, learned with a nonlinear and probabilistic technique. Segmentation is cast as a direct nonlinear minimisation of a level set-based, image-driven segmentation quality measure, with respect to the position in the latent space. Tracking is done using either a 2D or a 3D pose recovery stage. Along with segmentation and tracking, we can also simultaneously obtain generic sets of parameters (such as, for example, 3D articulated pose), by using joint shape-parameter spaces. Parts of work have been published in Prisacariu and Reid (2011a) and Prisacariu and Reid (2011c).

5.1 Introduction

In Chapter 3 we used 3D models to capture 2D shape variance. This led to a method that was both robust to imperfect image information and had low processing times. A direct extension to non rigid objects is however very difficult, as any unconstrained search in the full high dimensional space (that is characteristic to articulated or deformable objects) is very likely to run into local minima. As discussed in the literature

review, one way to make such searches tractable is to constrain them with activity or class information. In Chapter 4 we did this in a discriminative manner, using the output of a multiclass classifier to select between various activity specific articulated hand configurations. This allowed us to introduce limited, faux, articulated object support to the PWP3D tracker, but it still very much was a stopgap solution, as it only allowed us to model the shape space *discretely*.

Our overall aim in this chapter is to track (2D and 3D) generic deformable shapes from single monocular sources. To do this, we first need to model activity or class specific *continuous* shape spaces. An effective way to achieve this is to use dimensionality reduction, as shown by, among others, Tsai et al. (2003) and Dambreville et al. (2008a). The most popular method in the literature for dimensionality reduction is PCA. This is however a linear method, which means that (i) many dimensions are often required to capture all the variance in the shape space and (ii) there is no principled way to determine how likely a latent point is to generate a valid shape. Some works do use nonlinear dimensionality reduction, in the form of Kernel PCA, but these are still not probabilistic and introduce additional parameters that need to be manually tuned (which can lead to overfitting).

In this work we use a nonlinear and probabilistic dimensionality reduction technique called *Gaussian Process Latent Variable Models* (GP-LVM) (Lawrence, 2005). This allows us to capture more variance with fewer dimensions (compared to PCA) and still have a principled way of accessing a per latent point variance (compared to PCA and KPCA). Also compared to KPCA, all the parameters in GP-LVM are learned instead of manually set and all the processing effort is put into the training, rather than the testing phase.

We want to learn spaces for all types of shapes, both 2D and 3D, captured either implicitly (as level set functions) or explicitly (via arrays of points). Training a GP-LVM space involves finding the values for the latent variables and the hyperparameters for a Gaussian Process that maps the latent variables to their high dimensional counterparts. This is done via a nonlinear minimisation which becomes slow and likely to run into local minima when working with very high dimensional

shape descriptors. Therefore, in order to keep inference and training tractable, we use Fourier-based transforms to compress the shape descriptors. For explicitly defined 2D shapes we use elliptic Fourier descriptors, and for implicitly defined 2D or 3D shapes we use the discrete cosine transform.

The next challenge is finding a way to use these spaces effectively in segmentation. One approach to this problem would be to perform an unconstrained segmentation and then project this into the space of legal shapes to find the closest permitted shape. However, a persistent theme in this thesis has been that we aim to avoid this pre-segmentation step, instead casting the segmentation problem as a search *within* the restricted shape space. We therefore cast the segmentation problem as a direct nonlinear minimisation of an image-driven, level set-based energy function with respect to the position in the latent space. We use the same pixel wise posterior energy function as in Chapters 3 and 4, to which we add a factor to account for the GP-LVM trained variance.

Tracking is cast as either a 2D or a 3D pose recovery problem, similar to the one presented in Chapter 3; i.e. we minimise the segmentation quality measure with respect to the 2D or 3D pose parameters. Pose recovery and segmentation can be run either jointly or separately. In the 2D case we solve simultaneously for pose and segmentation. However in the 3D case, as we need to impose 3D shape consistency, we perform an iterative coordinate descent by alternately optimising shape and pose.

Since our aim is to use only single monocular sources, in the 3D case, the shapes must first be projected down into the image. As we represent 3D shapes implicitly, one possible solution to this problem would be to compute the isosurface of the zero level of each 3D level set function and project it down in a manner similar to Chapter 3. The isosurface algorithm is however not differentiable analytically, which makes optimisation difficult and inefficient. Our novel approach to this problem is based on the observation that an implicitly defined 3D shape can be seen as the log odds transform of a probability field that indicates how likely a 3D voxel is to be inside or outside the embedded surface. This allows us to introduce a method that computes a 2D probabilistic occupancy map, in an analytically differentiable manner, from a

3D implicitly defined shape.

In Chapters 3 and 4 we simultaneously obtained 2D segmentation and 3D pose. There are often other sets of parameters that need to be inferred from the shape of the object, such as the 3D articulated pose when working with human shapes, or the point of fixation when working with shapes of eyes. A standard, discriminative, approach to this problem is to learn mappings between descriptors extracted from the shapes and the sets of parameters, like we have already done in Chapter 4. There are several disadvantages of such methods, mainly that they cannot easily model multimodality, since, under this model, a shape can only correspond to a single set of parameters. Also, these methods are discrete rather than continuous. This means, for example, that given a human silhouette, such a method can select between the joint angles available in the training set, but it can not interpolate between them. A solution to both these problems is to use a generative approach and capture the joint probability distribution between the sets of parameters and the shape descriptors. Such a method was proposed by Ek (2009), where he extends GP-LVM to what he calls *shared GP-LVM* (SGP-LVM), i.e. a model that captures the joint variance between multiple training sets, using a single low dimensional latent space. Applying this method to our problem, in this chapter, we learn joint shape-parameter spaces, which we call *shared shape spaces*. These can generate, from a single latent variable, both the shape and the corresponding set of parameters. Therefore, by combining such spaces with shape and pose optimisations, we can simultaneously obtain the shape, the pose and corresponding set of parameters for a given image of an object.

Multiple shapes may correspond to the same parameters. For example multiple 3D car shapes may correspond to the same 2D projection. This inherent multimodality is captured naturally by shared GP-LVM, as different latent points can generate the same shape, but with different values for the sets of parameters. However, finding all the modes remains difficult, the usual method being to use multiple random points in the latent space as initialisations for multiple searches. To make searching for these modes more efficient, we propose a novel approach which learns mappings from the shape space to a space that encodes the similarity between shapes. This allows us,

for any given shape generated by a latent point, to find other latent points the are likely to generate a similar shape.

Another difficult problem is choosing the dimensionality of the learned latent space. A small number of dimensions translates into easily explorable spaces, but sacrifices representational power. A large number of dimensions captures higher variance, but leads to spaces that are more difficult to explore. Instead of fixing a single dimensionality, we propose a simple but novel extension which is to learn hierarchies of dimensions. The latent space corresponding to each level of such a hierarchy generates, along with shapes and sets of parameters, coordinates in the latent space above it in the hierarchy. This approach allows us to achieve both faster convergence and higher accuracy, which we explain in Section 5.3.

Our method therefore has the following advantages over previous work:

- compared to methods that use PCA, such as Tsai et al. (2003), by using GP-LVM, we place no linearity assumptions on the latent space, which allows us to capture more variance with fewer dimensions;

- unlike non probabilistic methods (i.e. the ones that use PCA or KPCA), such as Tsai et al. (2003) or Dambreville et al. (2008a), each point in the latent space is modelled by a Gaussian, whose mean and variance are found in a principled manner; this directly yields a probability for any given shape allowing us to use the shape space as a prior within a Bayesian framework (which has the effect of guiding the search to shapes of high probability);

- using shared shared spaces, we can recover the generic sets of parameters *and*, simultaneously, the segmentation of the object; this is in contrast to Ek (2009) where a pre segmentation step was required;

- we address the issue of multimodality, that is a problem for much work in this area, with a novel solution that learns mappings between shape spaces and spaces that capture similarity between shapes within the training data set; using this approach we can often search for the modes of the learned spaces in a more direct manner than using the standard exhaustive strategy;

- by using level set embedding functions to represent shapes, we can model all types of shapes (2D or 3D, with and without holes, with sudden topology changes, etc.);
- using Fourier based transforms to compress shapes, we use over 1000 times less memory than the current state of the art of Dambreville et al. (2008a);
- we propose a novel solution for the problem of choosing a dimensionality for the latent space, by using a hierarchy of dimensions, which enables us to obtain higher accuracy and faster convergence, compared to standard, non hierarchical, approaches.

The remainder of the chapter is structured as follows: we present our learning phase in Section 5.2 and our inference phase in Section 5.3. We show results obtained by applying our method to several images, video sequences and types of parameters in Section 5.4. We conclude in Section 5.5.

5.2 Learning Shape Spaces

Our task is to learn a low dimensional embedding for a set of n shapes $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$, where the shapes can be either 2D and represented implicitly or explicitly, or 3D and represented implicitly. We cast this problem into a generative framework using Gaussian Process Latent Variable Models (GP-LVM) (Lawrence, 2005), and we seek the set of latent variables $\mathbf{X}_Y = [\mathbf{x}_{\mathbf{y}_1}, \ldots, \mathbf{x}_{\mathbf{y}_n}]$ that captures as much of the variance as possible from the observation space sampled by $\mathbf{Y}$. Along with the set of variables $\mathbf{X}$, GP-LVM also finds the hyperparameters for a Gaussian Process mapping from $\mathbf{X}$ to $\mathbf{Y}$.

This task can be generalised by considering multiple observation spaces, where one observation space consists of the set of shapes $\mathbf{Y}$ and the others contain generic set of parameters, capturing, for example, 3D articulated pose. For two observation spaces, this strategy is known as shared GP-LVM (SGP-LVM) (Ek, 2009). Denoting with

$\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$ the sets of parameters and with $\mathbf{X}_{YZ}$ the shared shape-parameter latent space, the graphical model for shared GP-LVM is shown in Figure 5.1.

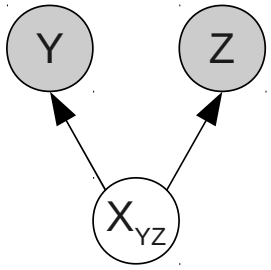


Figure 5.1: Shared GP-LVM graphical model. $\mathbf{Y}$ and $\mathbf{Z}$ are the observation spaces (the shape and the parameter spaces in our case) and $\mathbf{X}_{YZ}$ is the lower dimensional joint latent space.

Learning a shared space implies finding the joint low dimensional manifold for the $\mathbf{Y}$ and $\mathbf{Z}$ data sets. We do this by concatenating the observation points from the two data sets, $\mathbf{y}_i$ and $\mathbf{z}_i$ (with $i \in [1, \ldots, n]$), into $[\mathbf{y}_i, \mathbf{z}_i]$ pairs, thus building a new, joint data set $\mathbf{YZ}$. We then use the formalism of the standard (not shared) GP-LVM (detailed in Chapter 2) to find the low dimensional joint latent variables $\mathbf{X_{YZ}}$, and the hyperparameters of a GP mapping $\mathbf{X_{YZ}}$ to $\mathbf{YZ}$. For inference, as detailed in Section 5.3, we also need to be able to generate the shapes $\mathbf{Y}$ and the sets of parameters $\mathbf{Z}$ separately, so we also learn separate GP mappings from the joint latent space to each of the observation spaces.

For the remainder of this chapter, we only consider the case where multiple observation spaces are available, one for the shapes and the rest for the various sets of parameters.

As our covariance function we use the radial basis function kernel of Lawrence (2005):

$$\kappa(x_i, x_j) = \theta_1 e^{-\frac{\theta_2}{2}||x_i - x_j||^2} + \theta_3 + \theta_4 \delta_{ij} \tag{5.1}$$

where $\theta_k, k \in \{1, \ldots, 4\}$ are the hyperparameters and δ_{ij} is Kronecker's delta function.

In this formulation, GP-LVM does not guarantee that points that are close in the data space are also close in the latent space. This property is desirable, as it leads to a smaller number of local minima in our inference, which requires solving a nonlinear

optimisation problem, as discussed in Section 5.3. To preserve local distances we use the same method as Lawrence and Quinonero-Candela (2006) and Ek (2009), and represent the latent variables as smooth mappings (called back-constraints) from the observation spaces, and optimise w.r.t. the parameters of the back-constraints, rather than w.r.t. the latent points directly.

When a single observation space $\mathbf{Y}$ is available, formally, we can write:

$$x_{ij} = \sum_{j=1}^{n} w_{ij}\gamma(\mathbf{y}_i, \mathbf{y}_j) \tag{5.2}$$

where x_{ij} is the jth dimension of $\mathbf{x}_i$, i.e. $i \in [1, \ldots, n]$ and $j \in [1, \ldots, d]$, w_{ij} are the back-constraint parameters and γ is the back-constraining function. As in Ek (2009) and Lawrence and Quinonero-Candela (2006) our back-constraints are radial basis functions.

In the shared GP-LVM case, where, along with the shape space $\mathbf{Y}$ we also have a parameter observation space $\mathbf{Z}$, Equation 5.2 becomes:

$$x_{ij} = \sum_{j=1}^{n} w_{ij}\gamma(\mathbf{z}_i, \mathbf{z}_j) \tag{5.3}$$

As mentioned before, when learning the latent space, we optimise w.r.t. w_{ij} rather than directly w.r.t. $\mathbf{x}$.

Using an adequate shape representation (which we present in Section 5.2.3), GP-LVM allows us to learn nonlinear and probabilistic shape spaces. In Section 5.3.1 we show that it is possible to find a constrained segmentation of an object by minimising an image-based error directly in the learned latent space (so w.r.t. the latent space position). Section 5.3.2 shows how this segmentation can be linked to a 2D or a 3D pose recovery shape (similar to that of Chapter 3). This allows us, in the 3D case, to achieve simultaneous 2D segmentation, 3D pose recovery and 3D shape recovery. Going further, when multiple observation spaces (capturing various sets of parameters) are available, we can simply back-project the latent space position we found during segmentation to each parameter observation space to find the set of

parameters corresponding to the segmented shape. This process is detailed in Section 5.3.3.

There are however several issues which need to be addressed in order for us to be able to perform useful and timely inference. Firstly, choosing the number of dimensions of the latent space is often done empirically and dependent on the dataset. Secondly, while this model accounts for multimodality, searching for the modes is very inefficient. This issue is particularly important when doing parameter recovery. Thirdly, our system needs to be flexible to the type of shape used and allow for both 2D and 3D shapes and both explicit and implicit representations. This is due to the fact that explicit representations tend to lead to faster inference, while implicit ones allow for shapes of unconstrained topology (made out of single or multiple pieces, with and without holes). We address these issues in this section.

5.2.1 Hierarchy of Manifolds

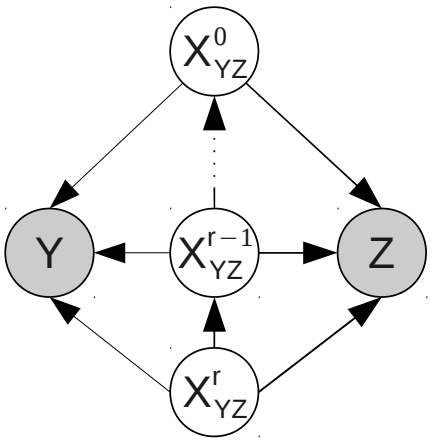


Figure 5.2: Graphical model for our hierarchy, when using two observation spaces. $\mathbf{Y}$ is the shape space, $\mathbf{Z}$ the parameter space and $[\mathbf{X}^0_{YZ}, \ldots, \mathbf{X}^r_{YZ}]$ the hierarchical shared shape-parameter spaces.

It is an open problem how best to choose the dimensionality of the latent manifold. A smaller number of dimensions leads to a denser latent space, with easier inference, but this may not capture the variability in the data adequately and consequently have less generative accuracy. A larger number of dimensions has greater generative

accuracy, but at the cost of a sparser latent space with more difficult inference. Almost all previous work (to our knowledge) chooses a unique dimension based on the amount of variance that has to be captured or on empirical observations about the data set. For example Navaratnam et al. (2007) use two dimensions, because the motion to be learned is known to be circular. In most cases however such a deduction is impossible. Furthermore, the optimum number of dimensions often differs for each point in the latent space. Assuming this optimum could be found for each point, the maximum optimum for the whole data set could be used as the target dimensionality. This is however likely to have a high value and therefore result in an inference that is slow and prone to local minima. Furthermore, for all the points that require a dimensionality smaller than the chosen one, this representation is wasteful both in speed and in memory.

Our novel solution to this dilemma is to learn a hierarchy of dimensions, as shown in Figure 5.2. Each $\mathbf{X}_{YZ}^{r}$ space is a lower dimensional embedding of the $\mathbf{X}_{YZ}^{r-1}$ space. $\mathbf{X}_{YZ}^{0}$ is the highest dimensional low dimensional embedding for the shared shape-parameter space manifold. We also learn GP mappings from each step of the hierarchy to each observation space. This approach enables us to generate an observation point directly from the each level in the hierarchy, without having to step through all its levels. Using these mappings, we can stop the inference as soon as the generated shape stop changing (between consecutive levels of the hierarchy). This leads to a considerable speed up of the inference, as shown in Section 5.4. In this work we use three hierarchy levels (i.e. $r = 3$): we first learn a mapping between the observation data and a 10D space, then between the 10D space and a 4D space and finally between the 4D space and a 2D space. We chose this set of dimensions (10D-4D-2D) empirically, as, throughout our tests: (i) searches in 2 dimensional spaces proved (usually) not to run into local minima, while still capturing a large amount of variance (as shown in Section 5.4); (ii) there was no discernible gain in the amount of variance capture when using more than 10 dimensions and (iii) 4 dimensional spaces captured approximately the average amount of variance between 2 and 10 dimensional spaces.

5.2.2 Manifold Modes Discovery

Multiple sets of parameters can correspond to a single shape. For example, multiple 3D car models can correspond to the same 2D car shape. Similarly, for human pose, multiple joint angle values can correspond to the same 2D silhouette. SGP-LVM is able to naturally capture this multimodality, as multiple latent points can map to the same shape but different sets of parameters. Therefore, the challenge that remains is finding all the latent points that generate any given shape. The standard approach to this task is to perform optimisation in the latent space by running several searches (minimisations of an energy function) initialised from multiple random points in the latent space. While obviously inefficient, this method is the only one to work on all types of data sets.

In many cases however, even though there are multiple modes in the mapping, there is a common source for the multimodality. For example, even though multiple 3D objects can project to similar 2D shapes, the ones that do often belong to the same class of 3D object. If a symmetric 3D object is rotated around its axis it produces similar 2D projections because of the symmetry. In a person's walk, it is the symmetry of the action that leads to similar silhouettes.

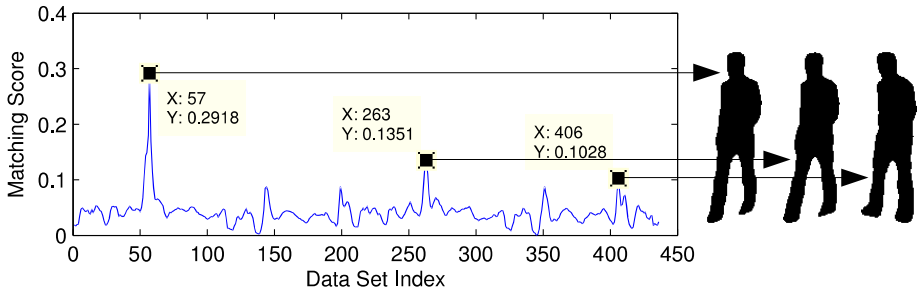


Figure 5.3: Example similarity function

To model the above-mentioned behaviour, in this work, we aim to learn a function that, for any given shape, tells us what are the other similar shapes in the training data set. Such a function is shown in Figure 5.3. For each shape in the data set, we compute a probability distribution describing the similarity between that shape and the others in the data set, using a standard similarity measure (in our case area

overlap). The number of sample points for this pdf is equal to the number of points in the data set. For larger data sets, approximations of this distribution could be used. For two shapes $\mathbf{y}_i$ and $\mathbf{y}_j$ in $\mathbf{Y}$ we can write this formally as:

$$s_{\mathbf{y}_i}(\mathbf{y}_j) = \frac{similarity(\mathbf{y}_i, \mathbf{y}_j)}{\sum_{k \in \mathbf{Y}} similarity(\mathbf{y}_i, \mathbf{y}_k)} \tag{5.4}$$

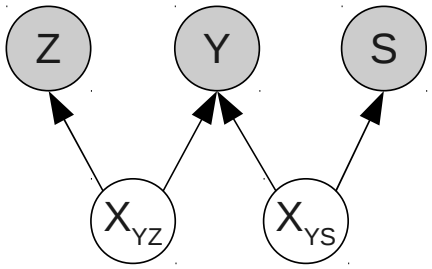


Figure 5.4: Graphical model for our modes discovery stage. $\mathbf{Y}$ is the shape space, $\mathbf{S}$ the similarity space, $\mathbf{X}_{YS}$ the shared space between shape and shape similarity and $\mathbf{X}_{YZ}$ the shaped space between shapes and sets of parameters.

We then learn a shared GP-LVM space between each shape and its corresponding $s_{\mathbf{y}_i}$. Figure 5.4 shows the graphical model for this part of the model. The joint space $\mathbf{X}_{YS}$ generates both the shape space $\mathbf{Y}$ and the similarity space $\mathbf{S}$. Thus, for any given latent point $\mathbf{x}_{ys}$ we can generate (i) a shape $\mathbf{y}_i$, and (ii) a function $s_{\mathbf{y}_i}$ over all shapes, whose modes are shapes that are similar to $\mathbf{y}_i$.

At inference time, to find the most similar shapes from the training data set, for a shape $\mathbf{y}$, we project the latent space point generating that shape to the space of similarity functions and find the peaks in that function, i.e. the set of indices i:

$$\{i \mid s_y(y_i) \text{ is locally maximum}\} \tag{5.5}$$

Figure 5.5 shows the results obtained by our method when used to find the modes in the space of shapes produced by rotating a person 360 degrees on a single axis. The point marked with the yellow circle in the $\mathbf{X}_{YS}$ space generates both a shape $\mathbf{y}$ (3rd row) and a function $s_{\mathbf{y}}$ (4th row). The top three peaks of this function indicate the top three points in the $\mathbf{X}_{YZ}$ latent space that are likely to generate similar shapes

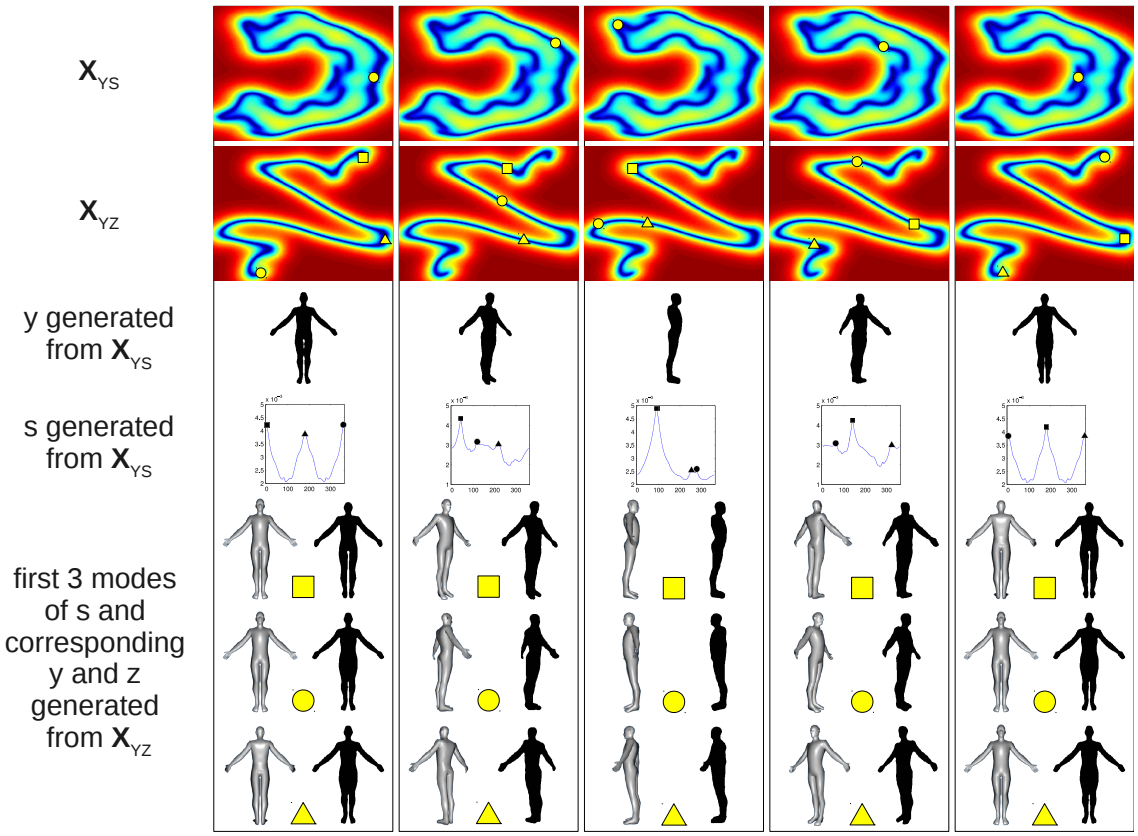


Figure 5.5: Example of our method discovering all the modes in the silhouette space of a person rotating by 360 degrees on a single axis. $\mathbf{X}_{YS}$ is the shared shape–similarity function space and $\mathbf{X}_{YZ}$ is the shared shape–pose space. Warmer colours (e.g. red) indicate higher variance, while colder colours (e.g. blue) indicate lower variance. The point marked in $\mathbf{X}_{YS}$ generates a shape $\mathbf{y}$ and a function $s_{\mathbf{y}}$ (3rd and 4th rows). The top three peaks of $s_{\mathbf{y}}$ indicate the top three $\mathbf{x}_{YZ}$ points that are likely to generate similar shapes (2nd row). The last three rows show the back-projections of the peaks to both shape and pose spaces.

(2nd row). The results of back-projecting these peaks to both the shape space and the pose space are shown in the last three rows.

5.2.3 Shape Representation

We need a differentiable and invertible shape descriptor. We also need to be able to work with both explicit and implicit representation of shapes. Explicit representations tend to be faster but cannot easily model the appearance and disappearance of holes, which can lead to discontinuities in the latent space (making inference more

difficult). Implicit representations, used extensively throughout the segmentation literature (Cremers and Funka-lea, 2006; Tsai et al., 2003), represent shapes with level set embedding functions (i.e. signed distance functions). These can deal with holes but lead to slower inference.

Due to their high dimensionality, when used with GP-LVM in their standard form, both these representations can make inference intractable and highly prone to local minima. This is especially obvious in the implicit case. Given a standard 640×480 image, a full signed distance function requires 307200 dimensions. Learning a 2D embedding space for such functions takes $\sim$8 hours and generating an embedding function from it takes $\sim$3 seconds.

In this work we compress both implicit and explicit shape representations using Fourier-based transforms. In the explicit case we use the elliptic Fourier descriptors of Kuhl and Giardina (1982), while in the implicit case we use the discrete cosine transform (Khayam, 2003). Note that the implicit representation also allows us to treat both 2D and 3D shapes identically (i.e. within the same framework), which is difficult in the explicit case, as explained below.

Explicit Shape Representation using Elliptic Fourier Descriptors

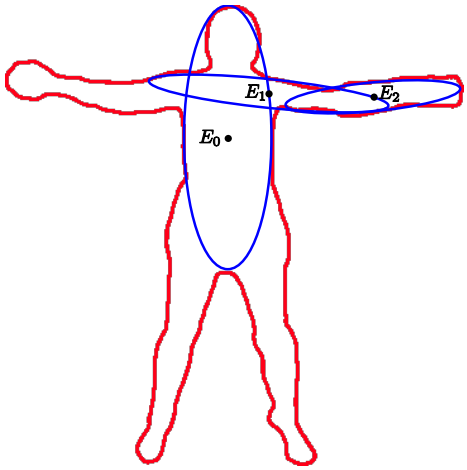


Figure 5.6: Example 3 harmonics of an elliptic Fourier descriptor.

Elliptic Fourier descriptors (Kuhl and Giardina, 1982) (EFDs) model shapes as sums of elliptic harmonics. Each harmonic is described by four coefficients, interpreted

geometrically as major axis length, minor axis length, orientation of the major axis and angle of phase of the ellipse.

As shown by Kuhl and Giardina (1982), any contour can be expressed as two periodic functions $x(t)$ and $y(t)$. For $x(t)$ this is written as:

$$x(t) = a_0 + \sum_{k=1}^{\infty} \left(a_k \cos \frac{2k\pi t}{T} + b_k \sin \frac{2k\pi t}{T} \right) \tag{5.6}$$

with T the perimeter of the contour, $t = 2\pi l/T$, l the arc length from a preset starting point, and:

$$a_0 = \frac{1}{T} \sum_{p=1}^{K} \frac{\Delta x_p}{2\Delta t_p} (t_p^2 - t_{p-1}^2) + \zeta_p (t_p - t_{p-1}) \tag{5.7}$$

$$a_k = \frac{T}{2k^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p} \left(\cos \frac{2k\pi t_p}{T} - \cos \frac{2k\pi t_{p-1}}{T} \right) \tag{5.8}$$

$$b_k = \frac{T}{2k^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p} \left(\sin \frac{2k\pi t_p}{T} - \sin \frac{2k\pi t_{p-1}}{T} \right) \tag{5.9}$$

with K being the number of points in the contour, $\Delta x_p = x_k - x_{k-1}$, $\Delta t_p = \sqrt{(\Delta x_p)^2 + (\Delta y_p)^2}$, and:

$$\zeta_p = \sum_{j=1}^{p-1} \Delta x_j - \frac{\Delta x_p}{\Delta t_p} \sum_{j=1}^{p-1} \Delta t_j \tag{5.10}$$

The term $y(t)$ is defined completely analogously in terms of the coefficients c_0, c_k and d_k.

EFDs require a starting point to be defined on the contour. Different starting points yield different descriptors, which leads to unwanted variance in the representation. We therefore need to first phase align the contours. This is straightforward in the 2D shape case but we discovered empirically that it does not scale well to the 3D case.

Implicit Shape Representation using Level Sets and Discrete Cosine Transforms

The discrete cosine transform (DCT) (Khayam, 2003) is a special case of the discrete Fourier transform, where the input data consists only of real numbers from an even function. It represents a function as a series of cosines, each with its own frequency and amplitude.

In the 2D case, the forward transform for an W-by-H image is:

$$D_{pq} = \alpha_p \alpha_q \sum_{m=1}^{W} \sum_{n=1}^{H} I_{mn} \cos \frac{\pi(2m+1)p}{2W} \cos \frac{\pi(2n+1)q}{2H} \tag{5.11}$$

with:

$$\alpha_p = \begin{cases} 1/\sqrt{W}, & p = 0 \\ \sqrt{2/W}, & p \in [1, W-1] \end{cases} \qquad \alpha_q = \begin{cases} 1/\sqrt{H}, & p = 0 \\ \sqrt{2/H}, & q \in [1, H-1] \end{cases} \tag{5.12}$$

D_{pq} are the coefficients of the DCT.

The inverse transform is defined as:

$$I_{mn} = \sum_{p=1}^{W} \sum_{q=1}^{H} \alpha_p \alpha_q D_{pq} \cos \frac{\pi(2m+1)p}{2W} \cos \frac{\pi(2n+1)q}{2H} \tag{5.13}$$

with $m \in [0, W-1]$ and $n \in [0, H-1]$.

Figure 5.7 shows an example shape and distance transform generated by our method from a 2D latent space of a star-jump motion. We also compute the actual (ground truth) distance transform of the contour. The chart shows vertical and horizontal cross sections in the per pixel difference between the two distance transforms. The maximum difference is smaller than 6 pixels. The average pixel error on all shapes in all our datasets was only *1.41 pixels*. Here we used $25 \times 25 = 625$ harmonics, whereas the full 640×480 embedding function would have required 307200 dimensions, so we achieve considerable compression in the dimensionality of the descriptor.

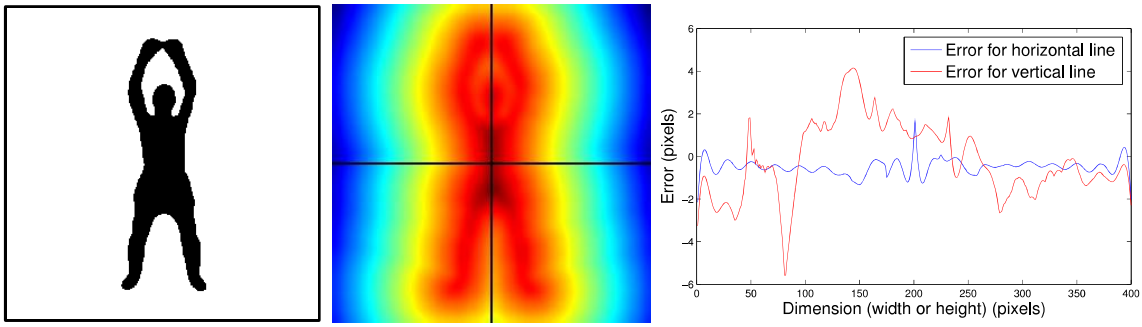


Figure 5.7: Distance transform accuracy example. The contour is the zero level of the middle distance transform, which is generated from a 2D latent space. The chart shows vertical and horizontal cross sections in the per pixel error between this distance transform and the ground truth one, for the same contour.

This representation also extends naturally to 3D shapes by replacing the 2D distance transform and discrete cosine transform with their 3D counterparts. When 3D distance transforms are used, however, they also need to be projected down to the image space. A naive method for doing this is to extract the zero level from the 3D level set function (using, for example, a marching cube algorithm) and to use this as a fixed 3D rigid model, in a manner similar to Chapter 3. In this chapter we introduce a more principled approach and define a projection function $\pi(\Phi)$, which takes a 3D distance transform and produces a 2D probabilistic image occupancy map, with value 1 "inside" the projection of the embedded 3D object and 0 "outside", with a smooth transition between the two.

To define $\pi(\Phi)$, we interpret the 3D distance transform Φ as the log odds transform of a probability field:

$$\Phi(l) = \log \frac{p_{inside}}{1 - p_{inside}} \tag{5.14}$$

with $l = (x, y, z)$ representing the 3D voxel locations inside the level set function and p_{inside} quantifying the probability of a location l being inside the closed 3D shape embedded by the level set function.

p_{inside} can then be extracted using the sigmoid function:

$$p_{inside}(l) = sigmoid(\Phi(l)) = \frac{\exp(\Phi(l))}{1 + \exp(\Phi(l))} \tag{5.15}$$

It then follows that, for any image pixel (u, v), we can define a p_{fg} as being the probability of (u, v) being the projection of a voxel from inside the 3D level set function:

$$p_{fg}(u, v) = 1 - \prod_{(l) \text{ on ray}} \Big(1 - p_{inside}(l)\Big) \tag{5.16}$$

with the product being computed for all 3D points that project to (u, v). Note that we assume pixel independence, which is not strictly true, as neighbouring pixels are much more likely to be members of the same class. Further improvements could be made to the projection function to take this fact into consideration.

For numerical stability we use the log space to compute this probability. We also introduce a parameter ζ to the projection function, to control the smoothness of the transition between the inside and outside regions. Therefore, our final projection function is:

$$\pi(\Phi) = 1 - \exp\Big(\sum_{l \text{ on ray}} \log\big(1 - \frac{e^{\Phi(l)\zeta}}{e^{\Phi(l)\zeta} + 1}\big)\Big) \tag{5.17}$$

The smoothness parameter ζ is constant throughout our tests, with a value of 0.75.

An example 3D model and its corresponding projection is depicted in Figure 5.8.

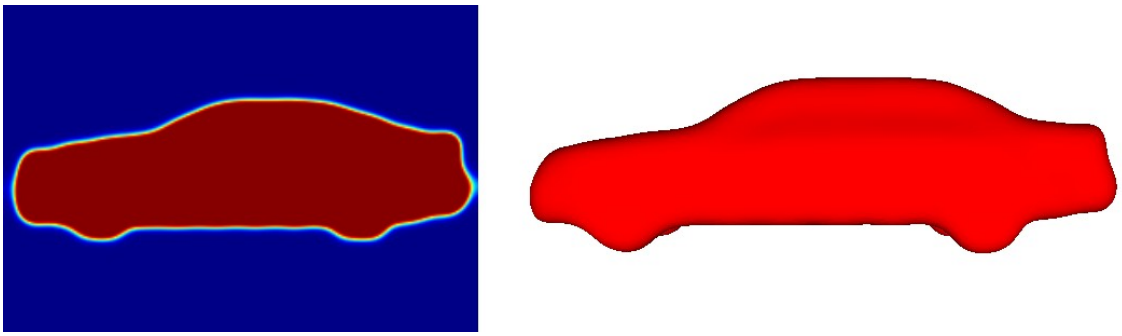


Figure 5.8: Example 3D model and projection: left – projection, blue represents $p_{fg} = 0$, red represents $p_{fg} = 1$; right – the 3D model that generated the projection.

5.3 Inference

The core of our inference is the minimisation of a level set based, image-driven energy function, with respect to a latent space point, which finds the low dimensional point

that generates a shape that best segments an image. We detail this minimisation in Section 5.3.1. A method for linking our shape optimisation with a pose recovery stage (i.e. tracking) is discussed in Section 5.3.2. Finally, our method for recovering generic sets of parameters simultaneously with pose and shape, is explored in Section 5.3.3.

5.3.1 Shape Minimisation

Let I be an image containing two disjoint regions, foreground and background, with known appearance models. We define a monotonic and continuously differentiable energy function $f(\Phi, I)$ as measuring how well the level set function Φ segments the image I. Φ can be either a 2D level set function or a 3D one that is first projected down to the image. Examples for $f(\Phi, I)$ are the Vese and Chan (2002) energy function or the Bibby and Reid (2008) energy function that has been used extensively in the previous chapters. Here $f(\Phi, I)$ is again based on the the Bibby-Reid energy function.

Typically, in level set-based segmentation, the contour evolves in an unconstrained space, i.e. $f(\Phi, I)$ is optimised by differentiating it with respect to time and using a standard nonlinear minimisation technique. In Chapter 3 we evolved the contour in the space of projections of the known 3D shape of the object we were trying to segment. Here we limit the evolution of the contour to the space of possible contours given by the learned latent space. We do this by differentiating $f(\Phi, I)$ with respect to the latent variables $\mathbf{x}_i, i = [1, \ldots, n]$. We begin by applying the chain rule:

$$\frac{\partial f}{\partial \mathbf{x}_i} = \frac{\partial f}{\partial \Phi} \frac{\partial \Phi}{\partial \mu} \frac{\partial \mu}{\partial \mathbf{x}_i} \tag{5.18}$$

The derivatives that are part of this chain rule expansion are detailed in the following three subsections.

Derivative of the GP-LVM Generative Process

In GP-LVM each latent space point $\mathbf{x}_i$ projects to a data point via a Gaussian distribution, with a mean μ and a variance $\upsilon = \sigma^2$:

$$\mu_i = K_* K^{-1} \mathbf{Y} \qquad \upsilon_i = \kappa(\mathbf{x}_i, \mathbf{x}_i) - K_* K^{-1} K_*^T \tag{5.19}$$

where K_* is the covariance between $\mathbf{x}_i$ and all the learned latent variable points.

$\frac{\partial \mu}{\partial \mathbf{x}_i}$ and $\frac{\partial \upsilon}{\partial \mathbf{x}_i}$ can be written as:

$$\frac{\partial \mu}{\partial \mathbf{x}_i} = \frac{\partial \kappa(\mathbf{x}_i, \mathbf{X})}{\partial \mathbf{x}_i} K^{-1} \mathbf{Y} \qquad \frac{\partial \upsilon}{\partial \mathbf{x}_i} = \frac{\partial \kappa(\mathbf{x}_i, \mathbf{x}_i)}{\partial \mathbf{x}_i} - 2 \frac{\partial \kappa(\mathbf{x}_i, \mathbf{X})}{\partial \mathbf{x}_i} K_*^T K^{-1} \tag{5.20}$$

In our case $\kappa(\cdot, \cdot)$ are effectively radial basis functions, so $\frac{\partial \kappa(\mathbf{x}_i, \mathbf{X})}{\partial \mathbf{x}_i}$ are arrays of differentials of the radial basis function, with respect to $\mathbf{x}_i$, for each learned latent space point.

Derivative of Φ w.r.t. the GP-LVM Mean

Φ is always a distance transform. It can however be:

- a 2D distance transform computed from an explicit contour (compressed using an EFD);
- a 2D or a 3D distance transform generated directly from the GP-LVM space;

Therefore, one formulation for $\frac{\partial \Phi}{\partial \mu}$ is required in the explicit case and another for the implicit one.

For the explicit case in which we represent 2D shapes using elliptic Fourier descriptors, we can write:

$$\frac{\partial \Phi}{\partial \mu} = \frac{\partial \Phi}{\partial x} \frac{\partial x}{\partial \mu} + \frac{\partial \Phi}{\partial y} \frac{\partial y}{\partial \mu} \tag{5.21}$$

where x and y are defined in Equation 5.6. This is an approximation of the true evolution of the level set with respect to its embedded contour, which is not continuous and therefore not analytically computable. When evaluated only in a small band

around the contour, this chain rule expansion is a correct approximation for a slow evolution of the embedding function, with the mention that $\frac{\partial x}{\partial \mu}$ and $\frac{\partial y}{\partial \mu}$ are defined only for the points on the actual contour. We approximate the derivatives for all the other points with those from their respective closest contour points. $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$ are computed using finite differences.

μ is an elliptic Fourier descriptor, so:

$$\mu = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ \dots & \dots & \dots & \dots \\ a_{n_a} & b_{n_a} & c_{n_a} & d_{n_a} \end{bmatrix} \tag{5.22}$$

We set a_0 and c_0, as defined in Section 5.2.3, to 0, in order to normalise the descriptor with respect to translation.

We expand $\frac{\partial x}{\partial \mu}$ as an array of $n_p \times n_a \times 4$ elements, $\frac{\partial x_t}{\partial a_k}$, $\frac{\partial x_t}{\partial b_k}$, $\frac{\partial x_t}{\partial c_k}$, $\frac{\partial x_t}{\partial d_k}$ with $t = [1, \dots, n_p]$, $k = [1, \dots, n_a]$, and:

$$\begin{aligned} \frac{\partial x_t}{\partial a_k} &= \frac{\cos 2k\pi t}{T} & \frac{\partial x_t}{\partial b_k} &= \frac{\sin 2k\pi t}{T} \\ \frac{\partial x_t}{\partial c_k} &= 0 & \frac{\partial x_t}{\partial d_k} &= 0 \end{aligned} \tag{5.23}$$

Similarly for $\frac{\partial y}{\partial \mu}$.

The implicit case is in fact simpler and more general, as the derivative of the inverse DCT of a matrix is the inverse DCT of the derivative of that matrix. This is straightforward to prove. For example, looking back at Equation 5.13 and limiting ourselves to the 1D DCT case, the inverse DCT formula is:

$$I_m = \sum_{q=1}^{W} \alpha_p D_p \cos \frac{\pi(2m+1)p}{2W} \tag{5.24}$$

D_p is a function of the latent variables x_i but neither α_p nor p are functions of x_i. We can therefore write:

$$\frac{\partial I_m}{\partial x_i} = \sum_{q=1}^{W} \alpha_p \frac{\partial D_p}{\partial x_i} \cos \frac{\pi(2m+1)p}{2W} \tag{5.25}$$

In our case this means that, since Φ is the inverse DCT of μ, $\dfrac{\partial \Phi}{\partial \mu}$ is just the inverse DCT of $\dfrac{\partial \mu}{\partial \mathbf{x}_i}$. This observation allows us to compute $\dfrac{\partial \Phi}{\partial \mu}$ using the fast inverse Fourier algorithm, with a complexity of O(N log N).

Definition of $f(\Phi, I)$ and derivative w.r.t Φ

For $f(\Phi, I)$, we use a level set formulation similar to the one presented in Chapter 3:

$$f(\Phi, I) = -\log(P(\Phi|\Omega, \mathbf{x}_i)) \tag{5.26}$$

where:

$$P(\Phi|\Omega, \mathbf{x_i}) = \prod_{x \in \Omega} \left\{ \pi\Big(\Phi(x)\Big) P_f + \Big(1 - \pi\big(\Phi(x)\big)\Big) P_b \right\} P(\Phi|\mathbf{x}_i) \tag{5.27}$$

Like before, Ω is the image domain, x is the pixel in the image, and P_f and P_b quantify the similarity between the pixel colour and the known foreground and background models.

We added a term containing the GP-generated variance υ (a function of the latent space position), which contains information regarding the validity of the generated shapes i.e. low variance points are more likely to generate valid shapes than high variance points. This term is written formally as:

$$P(\Phi|\mathbf{x}_i) = \frac{1}{\sigma\sqrt{2\pi}} \tag{5.28}$$

Effectively, this term is probability of the mean of the Gaussian distribution corresponding to the latent space point $\mathbf{x}_i$. Due to this term, we might not always get the shape best fitting the image, even if the latent space is actually able to generate

it, but rather the shape with the highest latent space probability. To account for this, after minimising the energy function with this term, we run another 10% of the total number of iterations without it.

$\pi(\Phi)$ is either the projection function detailed in Section 5.2.3 (when working with 3D shapes), or the smooth Heaviside function $H_e(\Phi)$ (when using 2D shapes).

Therefore, for our choice of $f(\Phi, I)$, $\frac{\partial f}{\partial \Phi}$ is:

$$\frac{\partial f}{\partial \Phi} = -\sum_{x \in \Omega} \frac{\partial \pi(\Phi)}{\partial \Phi} \frac{(P_f - P_b)}{\pi(\Phi)P_f + \big(1 - \pi(\Phi)\big)P_b} \frac{\partial \Phi}{\partial \mu} + \frac{1}{2\upsilon} \frac{\partial \upsilon}{\partial \mathbf{x}_i} \tag{5.29}$$

with $\frac{\partial \pi(\Phi)}{\partial \Phi}$ being either $\delta(\Phi)$, in the 2D case, or, in 3D case:

$$\frac{\partial \pi(\Phi)}{\partial \Phi} = \exp(...) \sum_{l \text{ on ray}} \frac{e^{\Phi(l)}}{e^{\Phi(l)} + 1} \tag{5.30}$$

where $\exp(...)$ is as defined in Equation 5.17 and l defined in Section 5.2.3.

5.3.2 Pose Minimisation

Our latent spaces only capture the variance of the space of shapes (and, optionally, joint with a set of parameters). Therefore, in order to track moving objects, we must also recover the 2D/3D pose. We do this by computing the derivatives of the Equation 5.26 with respect to the pose parameters in a manner similar to Bibby and Reid (2008) (in the 2D case) or Chapter 3 (in the 3D case).

For example, in the 3D case, using the notation from Chapter 3 to denote the pose parameters (λ_i with $i \in \{1, \ldots, 7\}$), we write:

$$\frac{\partial f}{\partial \lambda_p} = \frac{\partial f}{\partial \Phi} \left(\frac{\partial \Phi}{\partial x} \frac{\partial x}{\partial \lambda_p} + \frac{\partial \Phi}{\partial y} \frac{\partial y}{\partial \lambda_p} + \frac{\partial \Phi}{\partial z} \frac{\partial y}{\partial \lambda_p} \right) \tag{5.31}$$

where $\frac{\partial \Phi}{\partial x}$, $\frac{\partial \Phi}{\partial y}$ and $\frac{\partial \Phi}{\partial z}$ are computed numerically and $\frac{\partial x}{\partial \lambda_p}$, $\frac{\partial y}{\partial \lambda_p}$ and $\frac{\partial z}{\partial \lambda_p}$ are identical to the ones presented in Chapter 3.

The two sets of derivatives (with respect to latent space point and to pose) can be

used either as part of a joint optimisation towards the latent space position and the pose, or as part of a coordinate descent i.e. alternately iterating between the two sets of parameters. The most principled method is to do a single, one shot, optimisation, of both pose and shape. In our experience, this leads to slower convergence. Optimising pose until convergence and then shape until convergence is faster, as each separate optimisation has a smaller search space, compared to the combined one. However, finding the correct pose for the initial shape does not necessarily make it the best pose for the actual shape, and optimising for the latent space point with the wrong pose will often result in an incorrect shape.

In practice, in the 2D case, we noticed that both methods yield similar results, as shown in Section 5.4. In the 3D case however, it is often important to maintain shape consistency. When multiple frames are available, the shape of the object should be consistent among all the frames. The naive, adhoc solution to this problem is to choose a single informative frame, find the shape in this frame and use this shape to recover the pose (and implicitly the segmentation) in all the other frames. This is not often a good strategy, as we are unlikely to find a single frame that completely disambiguates the 3D shape, even when using a latent space shape prior.

A common solution to this problem, used throughout the 3D reconstruction literature, is to alternately iterate shape and pose optimisations. We take a similar approach, by alternating between shape iterations (over all the frames) and pose iterations (for each frame separately). We can perform joint optimisation (and indeed have done so), but our experiments suggest that separating these into alternation between pose and shape has no penalty in terms of accuracy, and confers the convenience that we can impose fixed shape over over many frames rather more conveniently.

In the 3D case, therefore, when a new frame is available, we proceed as follows:

- Iterate pose for the new frame using the shape approximation from the previous frame.
- Repeat until convergence:
  - Iterate the q shape parameters, over all frames.

– Iterate the 7 × number of frames pose parameters over all frames, using the new shape.

5.3.3 Parameter Recovery

To recover generic sets of parameters we first use the shared similarity space $\mathbf{X}_{YS}$ and the GP mapping from $\mathbf{X}_{YS}$ to $\mathbf{Y}$ to search for the latent space point that generates the shape $\mathbf{y}$ that best segments the image.

Next we project that point to the similarity space $\mathbf{S}$, using the GP mapping from $\mathbf{X}_{YS}$ to $\mathbf{S}$, to obtain a similarity function s.

We select the top n_p peaks in the similarity function, $[i_1, \ldots, i_{n_p}]$, as per Equation 5.5. n_p is a tunable parameter, which for our examples we have empirically set to 3. While better methods for sampling s might exist, this is not the focus of the present work.

Each index i corresponds to a latent variable $\mathbf{x}_i$ from the space at the r-th level of our hierarchy, denoted by $\mathbf{X}^r_{YZ}$. Next, we use $[\mathbf{x}_i, \ldots, \mathbf{x}_{i_{n_p}}]$ as initialisations for n_p other searches, this time in the $\mathbf{X}^r_{YZ}$ latent space, again aiming for an optimal (shape-constrained) segmentation of I.

The n_p results of these searches will most likely project to very similar shapes but different values for the parameter space $\mathbf{Z}$. They also project to points in the higher dimensional $\mathbf{X}^{r-1}_{YZ}$ latent space. These projections serve as initialisations for yet another n_p searches, this time in the $\mathbf{X}^{r-1}_{YZ}$ latent space, again looking to segment the image I. We repeat this operation for each latent space in the hierarchy.

The final result is a set of n_p possible segmentations for the image, each projecting to a separate set of parameters, generated from the highest dimensional latent space $\mathbf{X}^0_{YZ}$.

5.4 Results

In this section we evaluate the performance for the segmentation, pose and parameter recovery aspects of our method.

We begin in Section 5.4.1 with segmentation results. Here we capture both class and activity variability with both implicitly and explicitly represented 2D shapes. We also showcase the benefits of using shape priors and show several examples of our method dealing with large amounts of occlusions. Furthermore, we compare the generative accuracy of the PCA method of Tsai et al. (2003), the Kernel PCA method of Dambreville et al. (2008a) and our method using the implicit and explicit 2D shape representations. Finally, we quantify the speed and accuracy benefits brought by using the hierarchy of dimensions.

Section 5.4.2 shows pose recovery results. We show both 2D and 3D tracking results, focusing mostly on 2D/3D car shapes. We also show quantitative and qualitative proof that imposing 3D shape consistency is both beneficial and necessarily.

Section 5.4.3 shows various parameter recovery results, obtained by using our shared shape spaces approach. We learn three example joint spaces: a space linking car side view silhouette to 3D car shape, a space linking human 2D silhouette to human articulated pose and a space linking eye sclera shape to the point of fixation. Note that using 3D shape as a generic set of parameters is different from using it in pose recovery, since the 3D shape is never actually being used in the segmentation process. In the context of human pose recovery we also show a qualitative comparison between our method and several others in the literature using the Poser data set (Agarwal and Triggs, 2004) and the HumanEva I data set (Sigal et al., 2010), and obtain results that are at or close to the current state of the art.

The results section ends with a discussion on several possible failure cases, in Section 5.4.4.

5.4.1 Segmentation

We begin with examples which focus on finding and using a lower dimensional representation of 2D shape kinematics. The first example is a star jump action, for which we show the 2D latent space, together with samples from it and from the training data, in Figure 5.9. Notice that, even though we only use a 2 dimensional latent space, we can still keep most of the variance in the original data, and generate

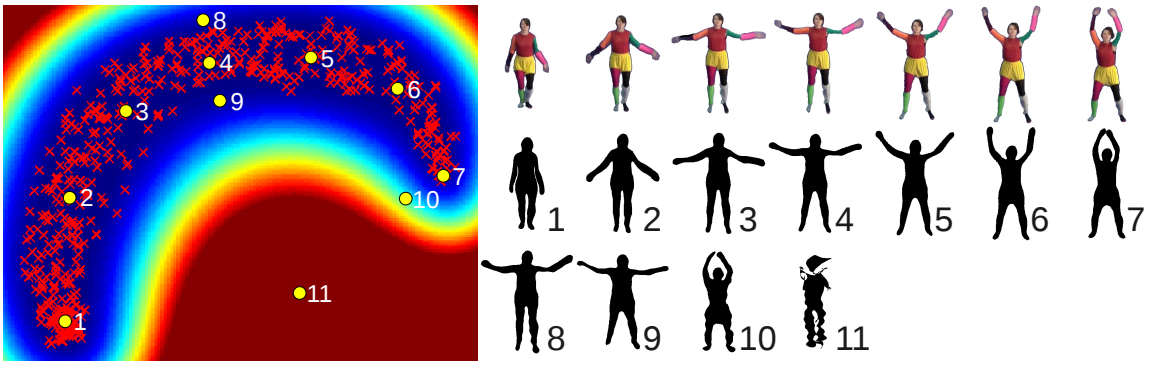


Figure 5.9: 2D latent space capturing the 2D shape kinematics of a person performing a star jump (left) together with samples from the training data (right, top row) and the latent space (right, bottom two rows). Warmer colours (e.g. red) indicate higher variance, while colder colours (e.g. blue) indicate lower variance. Samples 1–9 are taken from a low variance region (making them more likely to generate valid shapes) while 10 and 11 are from taken from higher variance regions (making them less likely to generate valid shapes).

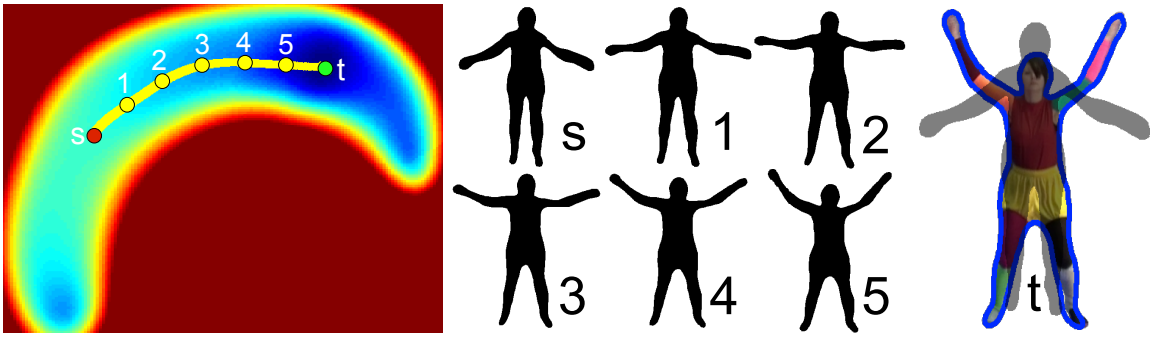


Figure 5.10: Example convergence of our method: left – values of the function for meaningful points in the latent space; middle – samples from the convergence trajectory; right – result (surrounded with blue) overlayed with initialisation (transparent black). Minimisation starts at the point marked with s and ends at the point marked with t.

accurate contours.

A typical convergence for our system for the star jump latent space is depicted in Figure 5.10. Here we evaluated our energy function at each meaningful point in the latent space and plotted the convergence trajectory through it, starting from the point marked with s and in red and ending with the point marked with t and in green. Notice that the energy function has a well defined minimum which we are able to reach even from a distant initialisation.

The quality of the segmentation is dictated by our ability to generate accurate

contours from the latent space. Obviously, accuracy increases with the number of dimensions of the latent space, as shown in Figure 5.11. A larger number of dimensions will however result in sparser latent spaces, making optimisation more difficult.



Figure 5.11: Impact of latent space dimensionality. A 4D latent space (b) leads to a better segmentation than a 2D latent space (a).

The previous examples focused on finding and using a lower dimensional representation of silhouette kinematics. Another typical scenario in introducing shape information to the segmentation process is using dimensionality reduction to capture the variance of an object class, for instance the variance in the space of all possible cars, planes, etc.

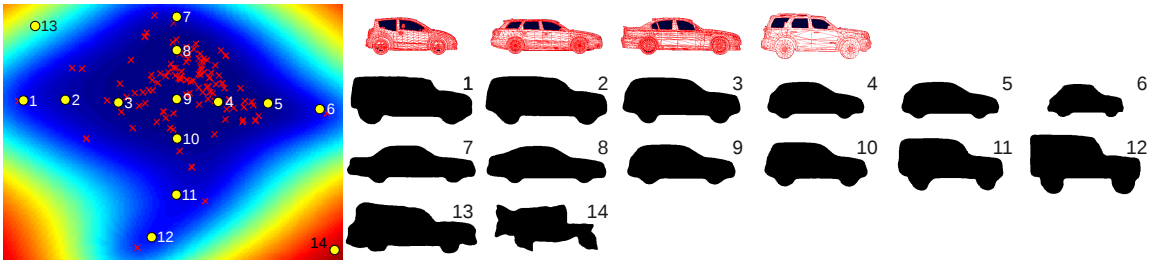


Figure 5.12: Example 2D latent space of side view car silhouettes (left), together with training data examples (right, top row) and latent space samples (right, bottom three rows). Samples 1-12 are taken from a low variance region of the space (more likely valid shapes) while samples 13 and 14 are taken from a high variance region (less likely valid shapes).

Figure 5.12 shows an example 2D latent space for 2D car side view silhouettes. Our method is again able to generate very accurate contours from the latent space.

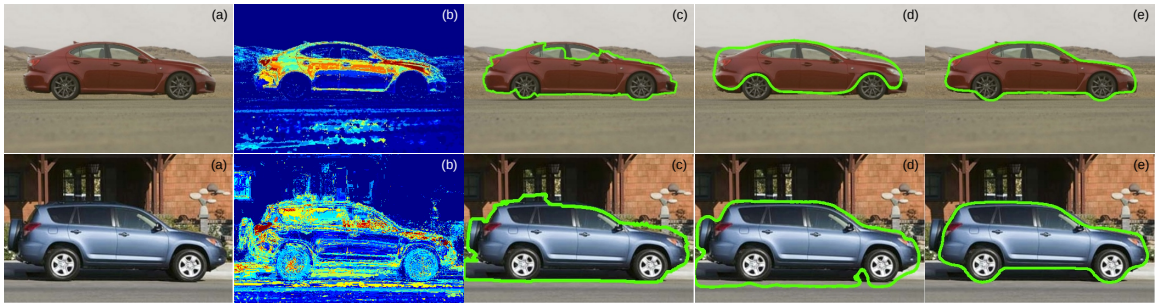


Figure 5.13: Example segmentations of cars: (a) – original images; (b) – map of $P_f - P_b$, where $P_f > P_b$; (c) – result using Grab Cuts (Rother et al., 2004); (d) – result using PWP (Bibby and Reid, 2008); (e) – our result.

Two examples of using the side view latent space to segment real images are shown in Figure 5.13. Here we compare our method to Grab Cuts (Rother et al., 2004) and the PWP level set segmentation of Bibby and Reid (2008), which do not use any prior shape information. We also depict the per pixel value of $P_f - P_b$, where $P_f - P_b > 0$ and P_f and P_b are defined according to Equation 3.5. In both examples our method produces more accurate results, in spite of working with heavily corrupted per pixel posterior probabilities.



Figure 5.14: Examples of our algorithm dealing with occlusions.

Perhaps the primary reason for introducing shape information in segmentation is trying to achieve robustness to occlusions. Figure 5.14 shows our algorithm dealing with high amounts of occlusion and still producing accurate results.

Next we show a study of generative and convergence accuracy. We measure accu-

racy using the Pascal VOC Challenge (Everingham et al., 2010) method i.e. divide the area of the intersection (of the test contour and the ground truth), with the area of the union.

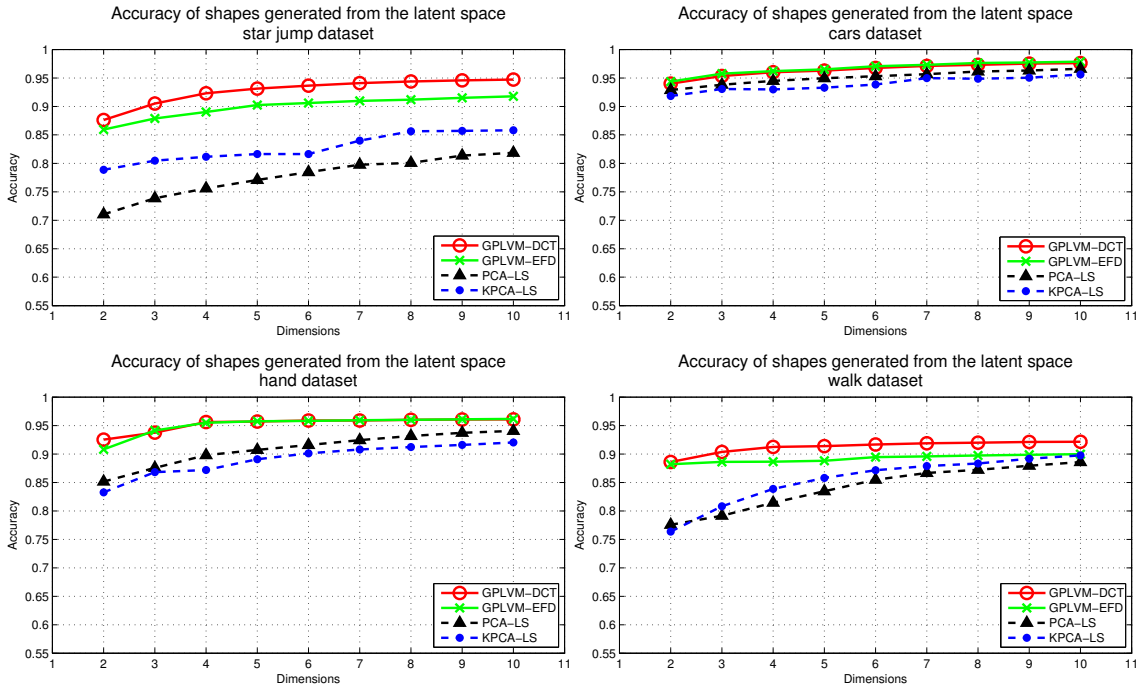


Figure 5.15: Accuracy comparison between our method using an implicit shape representation (red, circle marking), our method using an explicit shape representation (green, x marking), and the standard ones of using Kernel PCA (Dambreville et al., 2008a) (blue, dot marking) and PCA (Tsai et al., 2003) (black, triangle marking).

In Figure 5.15 we compare the generative accuracy of four methods used to learn latent spaces of shapes: ours using the implicit shape representation (GP-LVM – DCT) and the explicit shape representation (GP-LVM – EFD), the standard method of using PCA on the embedding functions (Tsai et al., 2003) (PCA – LS) and the Kernel PCA on the embedding functions method of Dambreville et al. (2008a), which is close to or at the current state of the art. Our method (with either representation) is more accurate than both Dambreville et al. (2008a) and Tsai et al. (2003). Compared to the PCA method of Tsai et al. (2003), our method is always capable of recovering more variance from fewer dimensions. Using a two dimensional latent space, we often produce similar (or better) results to a 10D PCA space. The difference in performance

is especially large when the shape deformations are highly nonlinear (the walking and star jump cases). Comparing the two types of shape representation, the implicit one proves constantly more accurate. The difference is largest when the data set contains large changes in shape topology, i.e. when the shapes break up and merge (the star-jump and the walk).

We also ran the generative accuracy test using GP-LVM directly on the embedding functions (without any Fourier compression) on the walk dataset, to measure the generative accuracy loss caused by our use of the Fourier-based descriptors. The results show a minimal ($\sim$1%) improvement over the explicit representation and an even smaller benefit over the implicit one ($\sim$0.2%). The main advantage of the compression is the computational time required. Direct use of the embedding functions is **very** time and memory intensive and is also more prone to local minima. For example, learning a 2D embedding function space takes $\sim$8 hours and generating an embedding function from it takes $\sim$3 seconds. In contrast learning a 2D elliptic Fourier space takes $\sim$20 minutes and generating descriptors from it takes $\sim$40 milliseconds. Similarly, learning a 2D DCT space takes $\sim$30 minutes and generating descriptors from it takes $\sim$80 milliseconds.

Convergence accuracy is tested by segmenting images for which we know the ground truth segmentation. Table 5.1 shows the convergence accuracy of our inference with and without the hierarchy of manifolds (using a 10D-4D-2D hierarchy). We learned latent spaces for a star jump, for 2D car side views, and for a segmentation of the sclera of a pair of eyes (with example shapes shown in Figure 5.25). The results depend on the intrinsic level of non-linearity in the space (i.e. the more a space is nonlinear, the more PCA dimensions are required to capture all its variance). A more nonlinear manifold is likely to lead to more local minima in the optimisation, as shown by the star jump example. The hierarchy helps explore the 10D space more extensively. When the manifold is more linear the hierarchy does not improve convergence accuracy by much. In all cases however, the hierarchy allows us to replace more costly 10D searches with less costly two and four dimensional searches, leading to average speed-ups of over 2.7x over running the optimisation in the 10D space

	10-4-2 Hierarchy	10D Alone	2D
Star jump	0.9373	0.9136	0.8438
Cars	0.9709	0.9676	0.9386
Eyes	0.8851	0.8837	0.8509

Table 5.1: Average convergence accuracy, with and without the hierarchy.

	10-4-2 Hierarchy	10D Alone	Speed-up
Star jump	48.56	153.69	3.16
Cars	41.71	107.11	2.56
Eyes	20.67	50.49	2.44

Table 5.2: Average convergence speed (seconds) per frame, with and without the hierarchy.

alone (Table 5.2). This speed-up is inverse proportional to the step size, but large step sizes can lead to divergence.

5.4.2 Pose Recovery

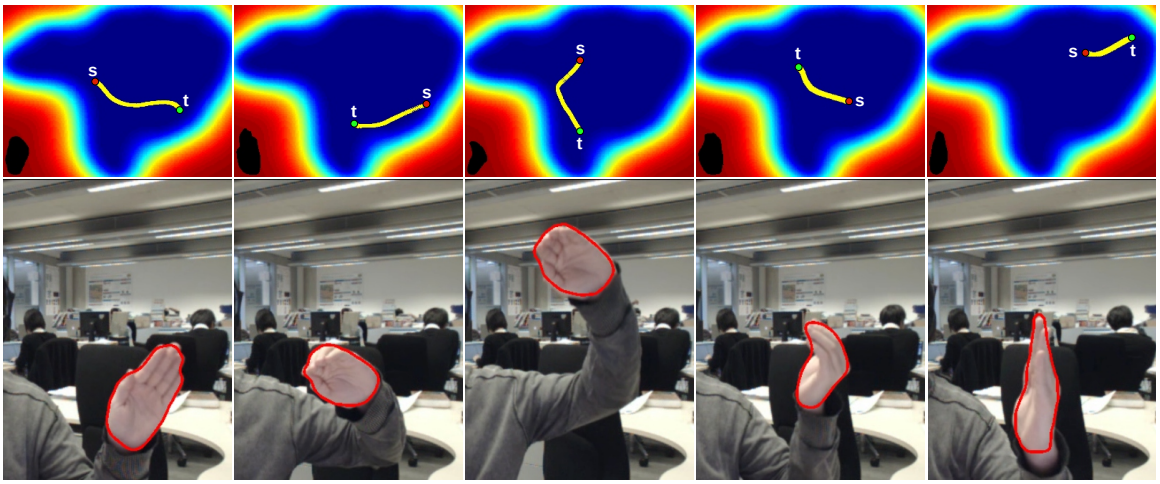


Figure 5.16: Tracking example. The latent space segmentation is combined with a 2D pose recovery step, with each minimisation being alternately iterated to convergence. The top row shows the latent space and convergence trajectory for each example frame. The starting shape is shown in black. The convergence trajectory starts from the s point and ends with the t point. The bottom row depicts the result.

As shown in Section 5.3.2, we can link our latent space-based segmentation with

a 2D or a 3D rigid pose recovery step. Figure 5.16 shows our algorithm tracking a deforming hand in 2D. The 2D pose recovery step keeps the hand centred, at a constant scale, and in the upright position.



Figure 5.17: Example convergence steps for the 2D pose / latent space one shot minimisation.

Figure 5.17 shows an example convergence of our joint 2D pose and 2D shape optimisation. Notice that even though the initial shapes are quite different and the initial pose is far from the correct one, the result is still correct.

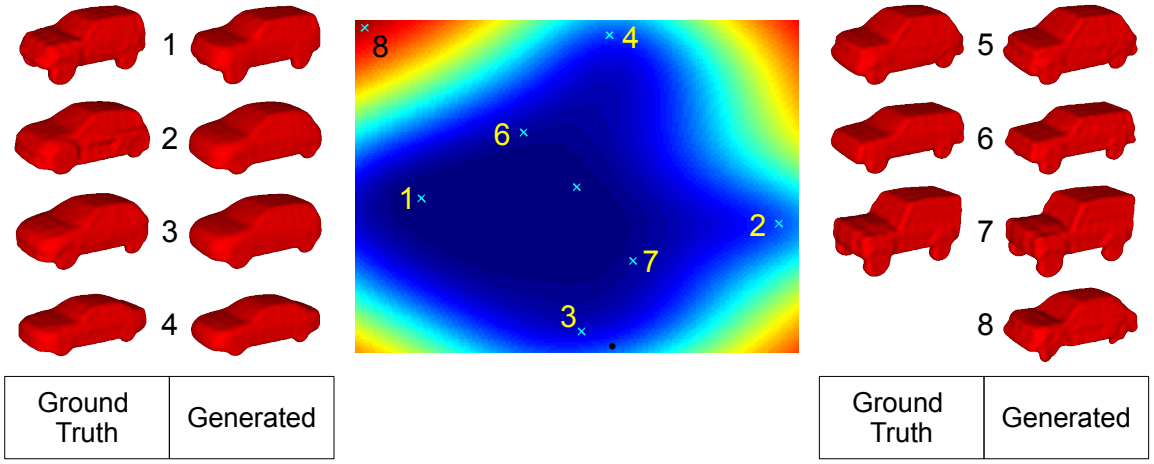


Figure 5.18: Example 2D latent space of 3D car shapes. Shapes 1-7 are points inside the low variance region of the space (in blue), while shape 8 is from the high variance area (in red). The sample from the high variance area does not have a corresponding ground truth because it was not part of the original training data.

Figure 5.18 shows an example 2D latent space generating 3D car shapes. A sample 3D pose recovery convergence is shown in Figure 5.19. Here we intentionally started far from the correct value to show the algorithm is able to converge from gross shape and pose errors. We adapted both shape and pose simultaneously and the algorithm converged in ∼300 gradient descent iterations. More powerful optimisation methods

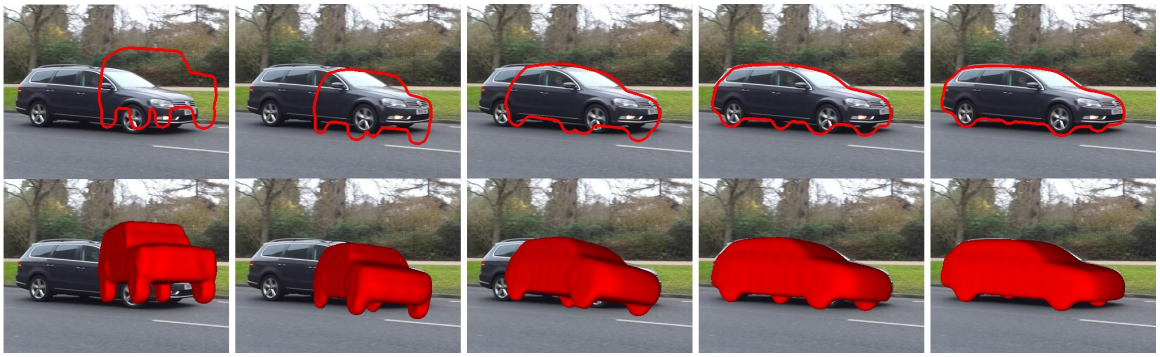


Figure 5.19: Example 3D shape and 3D pose convergence for our algorithm.



Figure 5.20: Film strips showing our algorithm successfully tracking different types of cars in different environments.

could potentially be used, as there are no mathematical impediments to computing a second derivative of our energy function (unlike the method presented in Chapter 3 which requires the derivative of the Dirac delta).

Two tracking results using the car latent space for two different types of car (sedan and hatchback), are shown in Figure 5.20. Both cars are being tracked successfully throughout their respective sequences.

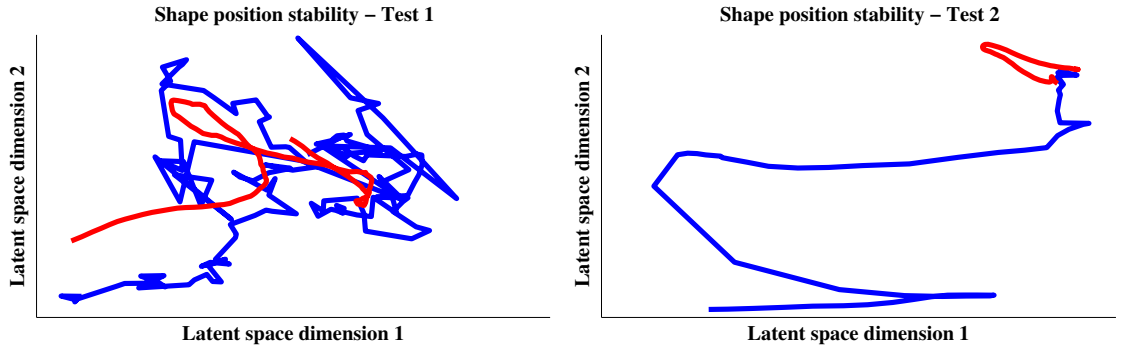


Figure 5.21: Charts showing recovered latent space positions, with (in red) and without (in blue) shape consistency. When shape consistency is used the trajectories in the recovered latent space are smoother and tighter.

Enforcing shape consistency is essential. Figure 5.21 shows two charts for two different video sequences, with Figure 5.20(bottom) corresponding to test 1 and Figure 5.21 to test 2. We ran our methods twice on each video sequence, once with and once without the shape consistency constraints. In both examples the result was smoother when shape consistency was used (red lines in Figure 5.21). Furthermore, in the second example the lack of shape consistency leads to a complete failure of tracking. As shown in Figure 5.22, when the cyclist occludes the car, the lack of shape consistency causes the algorithm to incorrectly adapt shape, ultimately resulting in failure. When shape consistency is enforced the algorithm can use information from unoccluded frames, which insures stable tracking.

5.4.3 Parameter Recovery

We have tested the parameter recovery performance of our method in three different fields: monocular 3D reconstruction, gaze tracking and 3D pose recovery.

First we show results in monocular 3D reconstruction. Here we learn joint latent spaces of car silhouettes (back and side views simultaneously) and 3D models (Figure 5.23). We run the segmentation using the 2D shape space and generate the corresponding 3D shape from the resulting latent space. Note that this is different from the 3D pose/shape recovery results from the previous section as the 3D models



Figure 5.22: Failure due to occlusion combined with lack of shape consistency. When the cyclist passes in front of the camera and shape consistency is not kept, the system fails (top two rows). When information from multiple frames is integrated the system successfully tracks through the occlusions (bottom two rows).

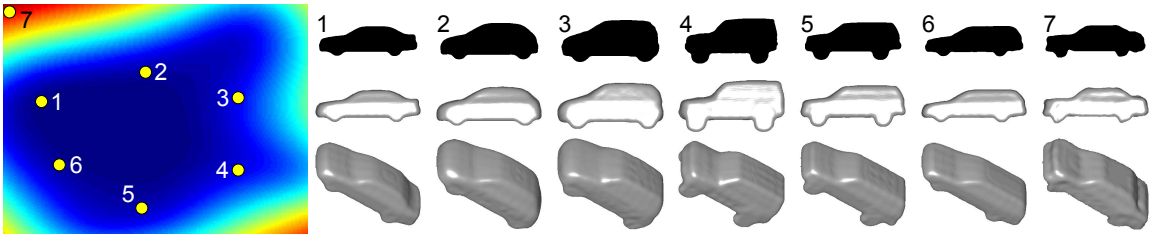


Figure 5.23: Example joint latent space (left) for side view car silhouettes and 3D models. Samples 1-6 are taken from a low variance region of the space (so more likely valid shapes) while sample 7 is taken from a high variance region (so less likely valid shapes).

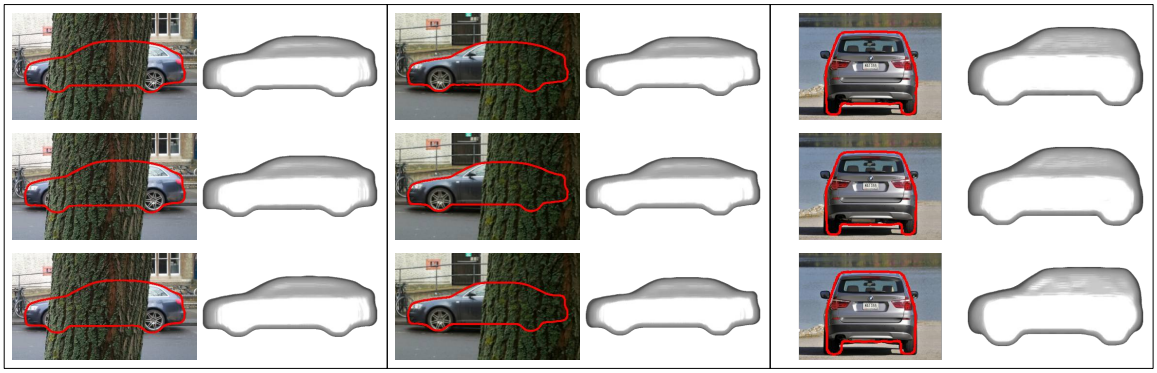


Figure 5.24: Examples of our algorithm correctly generating 3D car models from 2D car silhouettes.

are never used in the actual segmentation process. Figure 5.24 shows our algorithm working, in spite of heavy occlusions, and correctly finding the modes in the latent space. The left panel shows an image with no ambiguity in the shape. All three mode searches converge to the same 2D shape and 3D model. The middle panel shows an ambiguous case where the rear of the car is occluded. The three searches yield putative segmentations and corresponding 3D models that range from sedan to hatchback, all equally likely given the silhouette data. Finally, in the right panel, there are no occlusions, but the back view of the car is still ambiguous. Here the three modes represent different car types: a small hatchback (first two modes) and a SUV (third mode).

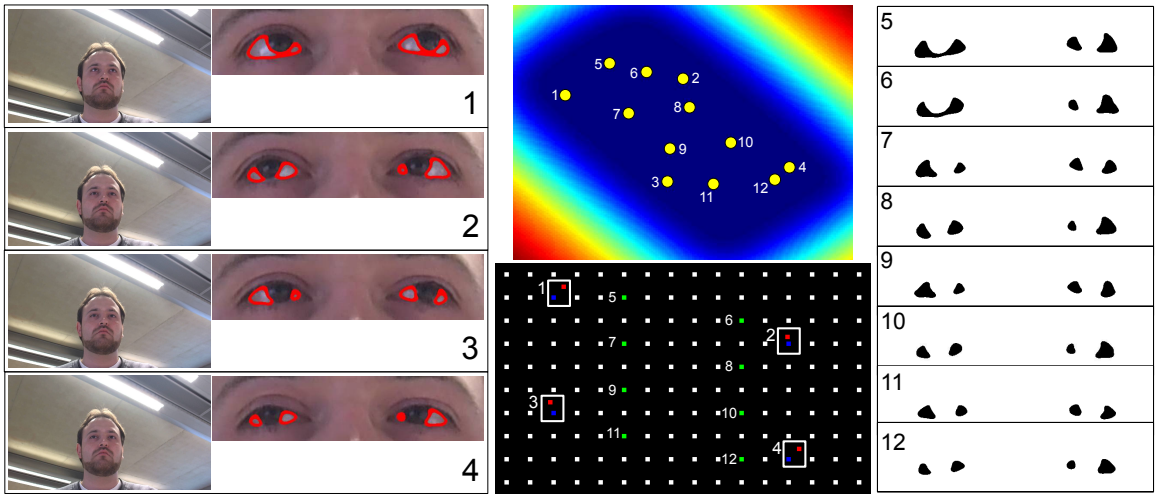


Figure 5.25: Gaze tracking. The latent space (middle column, top) generates both eye sclera shapes and location coordinates. To test and train we displayed a grid of dots (spaced at 32.23mm) on the screen (middle column, bottom) and looked towards each one. Images 5-12 show training shapes. Their respective screen locations are marked green on the grid. Images 1-4 show segmentation results. Positions 1-4 on the grid show the ground truth (blue) and the value generated by our algorithm (red).

Figure 5.25 shows our algorithm tracking gaze by segmenting the sclera of the eye. We learn a joint latent space between the shape of the sclera and point of fixation on the screen (middle column, top). To get this position we displayed a grid of 16×10 dots (spaced at 32.23mm) on the screen (middle column, bottom) and looked towards each point. Images 5-12 show example training shapes. Their respective points of

fixation are marked with green on the grid. Images 1-4 show segmentation results. Positions 1-4 on the grid show the ground truth (blue) and the value generated by our algorithm (red). Our method correctly and accurately segments all the parts of the sclera. The average error was 16.57mm or *1.58 degrees* visual angle error (the head was situated at around 60cm from the screen). While this error is bigger than the state of the art in gaze tracking with special lights and cameras, it is smaller than that of webcam-based gaze tracking (usually around 3 degrees).

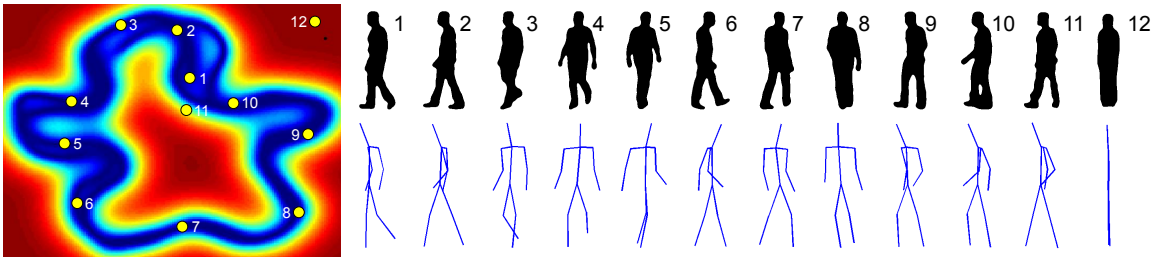


Figure 5.26: Example joint latent space (left) between 3D poses and 3D silhouettes for a walk from the HumanEva data set.

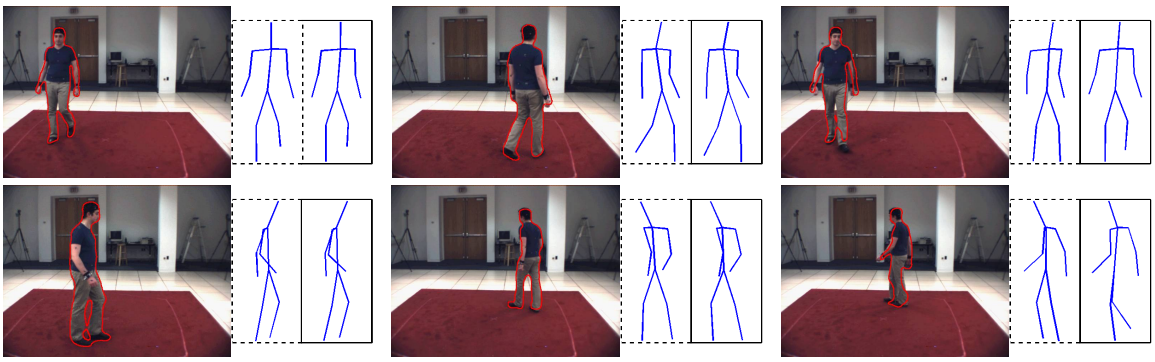


Figure 5.27: Example HumanEva results (4 accurate and 2 failure cases). Dotted line marks ground truth and filled line marks our results.

Perhaps the most obvious application of our method is pose recovery. We therefore compared to two standard data sets, the Poser one of Agarwal and Triggs (2006) and the HumanEva I one of Sigal et al. (2010). Here we used four dimensional latent spaces, same as Ek (2009), but we did not use any latent space dynamics. The Poser data set consists of 1927 training and 418 test images, all artificially generated from real mocap data. Table 5.3 shows that our method achieves state of the art

results. Here, of the three modes, we selected the one that produced the highest image segmentation score. HumanEva I consists of video sequences with synchronised mocap data, recorded for three subjects. Figure 5.26 shows samples from the joint pose–shape space, for a walk in HumanEva I. Figure 5.27 shows results on the same data set. The error in this data set is the average Euclidean distance between 15 3D joint locations, in camera coordinates. Since we do not recover 3D translation, we only provide results in local coordinates. This prevented us from computing the error on the actual test data from the data set (mocap information for this part is not included). We did test on the validation part of the data set (where mocap information is provided). The error was *28.10*mm, which would place our method close to state of the art in monocular 3D pose recovery. Dedicated pose recovery methods, with (marginally) better performance, do exist, such as Urtasun and Darrell (2008), but we emphasise that this is only an application of our more general framework, that is not restricted to pose recovery.

Algorithm	RMS Angle Error (degrees)
Linear Regression (Ek et al., 2007)	7.70
Nearest Neighbour (Sigal et al., 2009)	6.97
GPLVM (Ek et al., 2007)	6.50
Kernel Regression (Sigal et al., 2009)	6.03
Gaussian RVM (Agarwal and Triggs, 2004)	6.00
Global sKIE (Sigal et al., 2009)	5.95
Local sKIE (Sigal et al., 2009)	5.77
Shared Shape Spaces	5.25

Table 5.3: Performance of our algorithm on the Poser data set (Agarwal and Triggs, 2004)

5.4.4 Failure Cases

We show two failure cases of our method in Figure 5.28. In this case the specific type of car featured here (or anything close) did not appear among the training data points, so it cannot be generated accurately from a latent space. The results can be improved by introducing more training data. The second failure case is caused by

the high degree of occlusion and especially the occlusion of the two extremities of the car, which would have "stretched" the segmentation.



Figure 5.28: Failure cases: left – failure due to lack of training data; right – failure due to high degree of occlusion.

5.5 Conclusions

In this chapter we presented a method for finding nonlinear shape manifolds and using them as shape priors in level set based segmentation and tracking. Shapes can be 2D or 3D, represented implicitly or explicitly, made up by single or multiple parts and can have holes. Inference is made tractable by using Fourier-based transforms to compress the shape descriptors. Along with shape, we can also simultaneously recover generic sets of parameters. These are very versatile and can represent, for example, 3D pose, 3D shape or eye gaze. For many such applications we have shown results beyond or close to the current state of the art.

In developing our system we have also made three algorithmic and representational contributions that may also have applications in other probe domains. First, we have shown how the multimodal space can be more effectively explored by introducing the shared shape – shape-similarity space. Second, we have introduced a hierarchy of latent spaces in order to improve convergence and accuracy via a coarse to fine search. Third, we have proposed a projection function that creates 2D probabilistic occupancy maps directly from 3D level set functions.

Some open issues still require further research. Chiefly among these, GP-LVM cannot deal with large amounts of data. When using more than 3000 data points, learning and inference quickly become intractable. Several sparsification techniques exist and could prove useful. Alternatively, local spaces could be learned, provided a method for selecting between them could be developed.

Chapter 6

Conclusions

6.1 Summary

In this thesis we looked at how shape information is to be represented and used in segmentation and tracking, with a view towards making the two tasks simultaneous. Therefore our aim has been to find ways by which prior knowledge could be used to constrain the segmentation search space, while at the same time producing either 2D / 3D rigid pose or a generic set of parameters, or both. Furthermore, all our work is region based, which enables our methods to be robust to real world problems, such as cluttered environments, occlusion, motion blur, etc.

We began in Chapter 2 with a review of the current state of the art and found a dichotomy in the way shape knowledge is used in segmentation. One school of though assumes that the space of 2D shapes an object can take in an image is a function of its rigid 3D shape and 3D pose. Another one tries to learn the space of 2D shapes from examples. Both have their respective advantages, but unfortunately these are most often outweighed by their disadvantages. For example, the use of edges is very widespread in the literature, but this generally leads to methods that cannot work with cluttered environments, occlusions or motion blur, making them unsuitable for real world usage. Alternatively, when more complicated features are used (such as point features for example) good tracking performance is available, but the systems

require many tens of seconds per frame. The second category of methods usually assume the shape space to be linear. This leads to many dimensions being required to learn even simple spaces which, because of the high dimensionality, are difficult to search. Perhaps more importantly, the shape spaces are generally not used directly to drive the segmentation process, but rather, segmentations are extracted from the image and then are vetted by the space space.

In Chapter 3 we presented our first attempt at using shape knowledge, the PWP3D tracker. Here an object is represented by its explicit 3D shape. The 2D segmentation process is cast as a minimisation of a level set, region based segmentation quality measure inside the 6 degree of freedom pose space of the known 3D shape. Unlike all previous work in 3D tracking, we use posterior membership probabilities for foreground and background pixels, rather than pixel likelihoods, on which we also impose temporal consistency. We also extend the base algorithm to track multiple objects from multiple views and support our ideas with a real-time implementation.

In Chapter 4 we use our PWP3D tracker in two applications, also providing examples of how inherent problems of our 3D tracker can be solved. Firstly, it often happens that the positions of an object does not overlap between consecutive frames. In such cases PWP3D often loses tracking. We address this problem in our first application, a driver assistance system, where we use a detector to relocalise the tracked object at every frame. The second problem is multimodality. The mapping from silhouette to pose is multimodal, meaning that multiple different poses can lead to the object projecting to the same silhouette. Our second application, a 3D rigid hand tracker, tries to mitigate this problem by using data coming from an off-the-shelf accelerometer placed on the hand. Finally, our 3D tracker is limited to a single, rigid shape. In the second application of Chapter 4 we also show how a multi-class classifier could be used to switch between a subset of activity specific 3D shapes, effectively allowing for faux deformable object support.

Learning of shape spaces from examples is explored in Chapter 5. Like much of the current literature, we learn shape spaces using dimensionality reduction. Unlike any previous work however, we use a nonlinear and probabilistic dimensionality re-

duction technique, called Gaussian Process Latent Variable Models. This has several advantages, the most important being that (compared to PCA and Kernel PCA), we can use fewer dimensions to capture more variance. GP-LVM also allows us to access the per lower dimensional latent point variance in a principled manner. The image segmentation is driven directly from the learned latent space, and we impose that both the shape evolution and the final result have high probability. Also unlike any previous work, using an extension of GP-LVM called shared GP-LVM, we can learn joint shape – parameter spaces. Segmentation flows exactly like before, but once a lower dimensional latent point has been found, it can be projected to the parameter observation space, for the simultaneous recovery of both shape and parameter set. The parameter set can represent anything from 3D shape to 3D articulated pose (3D joint angles) and to eye gaze. We also bring two improvements to shared GP-LVM: first, we show how the multimodal space can be more effectively explored by introducing the shared shape - shape-similarity space and, second, we introduce a hierarchy of latent spaces in order to improve convergence and accuracy via a coarse to fine search. Lastly, our system is able to work with all types of shapes, 2D or 3D, represented either implicitly or explicitly, and, to make learning and inference tractable, we use Fourier based transforms to compress the shape descriptor. This results in a very large economy in memory and increase in speed. For 3D shapes for example, compared to the current state of the art, our method uses more than 1100 times less memory.

6.2 Future Work

Our contributions are generic in the fields of segmentation and tracking. Throughout this thesis, along with a focus on the theoretical aspects of our research, we have also shown several applications of our methods, in the form of a driver assistance system and an example human computer interface. Several other applications are also possible. Most significantly, in the field of medical imaging, we could use MRI and/or ultrasound instead of standard camera images. Since shared shape spaces

allow for joint shape-parameter information to be learned, applications such as learning mappings between ultrasound heart data and heart disease diagnostics may be possible.

Along with other applications, several potential algorithmic improvements could be made to the methods themselves. We split these in two categories: low level improvements and high level improvements.

6.2.1 Low Level Improvements

The most common source of failure for the methods present in this thesis is the foreground background discrimination given by the per pixel image statistics. The use of pixel wise posteriors instead of likelihoods has already lead to a considerable improvement in tracking performance, as shown in Chapter 3. They are still however far from perfect.

One possible improvement is to model global spatial information. Pixel wise posteriors, as they are based on histograms, discard all spatial information, losing valuable image information in the process. Modelling this information is challenging, since, as the pose or shape of an object changes, the locations of the colours in the image change. Any effective spatial information preserving colour model would therefore need to be defined in an object centric coordinate system. This could be achieved by learning "per surface" histograms, but this leads to a second problem, namely a much smaller basin of convergence. A solution for this would be some sort of hierarchy of details, the lowest level being a standard histogram and the highest level the full, original image. Matching would follow incrementally.

Local spatial information could also be modelled. This could be done by building models of image patches (centred on each pixel location), instead of just pixels. Going further, we would need to replace the simple histograms with some sort of multi class classifier, since the image information would be much more sparse and high dimensional. After some initial testing we have found that random forests would be a good choice and do indeed produce results superior to the original pixel wise posteriors (see Figure 6.1). The problem that remains to be investigated is how these

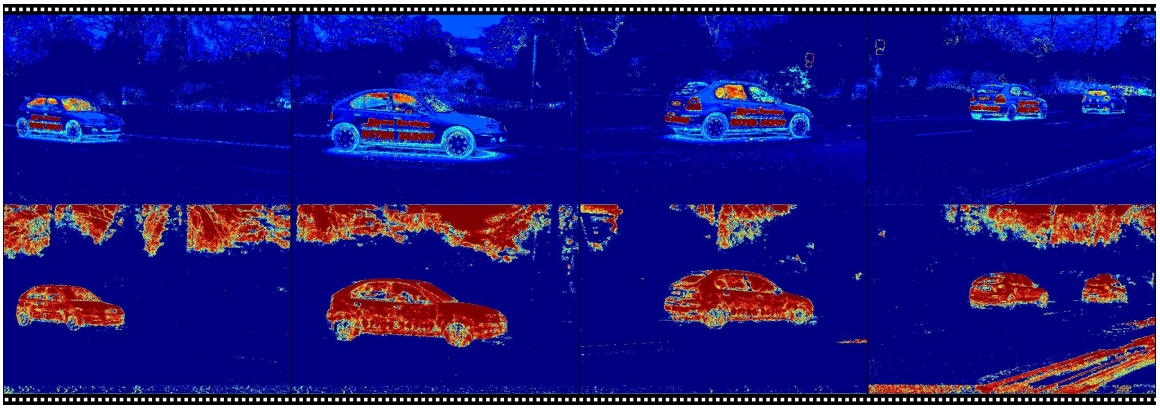


Figure 6.1: Comparison between the foreground / background separation provided by the PWP method and a RF classifier. Warm colours indicate high probability and cold ones low probability.

models could be learned quickly and online.

Though we have made some improvements to the online appearance model adaptations, allowing for more aggressive updating, we are still only using a single, global appearance model of the tracked object. More interestingly, one could take advantage of the fact that appearance of a 3D object is a function of pose, and we believe that (online) learning of such an appearance model may be both feasible and beneficial.

6.2.2 High Level Improvements

Throughout this thesis we have only used 2D contours to provide image information and have disregarded any feature inside these contours. The mapping from contour to 3D pose or 3D shape is unfortunately ambiguous and multimodal i.e. multiple 3D shape-3D pose pairs will project to the same contour.

Our shared shape formulation could easily be extended to incorporate the information inside the contour, by defining multiple "views". For example, given the space of 3D car shapes, one view could be the side silhouette, another view could be the top silhouette and a third view could be a descriptor encoding markings present on the car. Or, given a space of human walking poses, one view could be a silhouette while another view could be a descriptor for the edges inside the silhouette.

When the ambiguity between 3D pose/shape and 2D shape cannot be explained

away by more complex image information, multiple hypothesis about the pose/shape should be maintained. We have already done this in a limited manner when we learned similarity spaces in Chapter 5. A direction of research would involve using a full particle filter inside the lower dimensional latent space.

To represent explicitly defined shapes two one dimensional Fourier descriptors are used. For implicitly defined shapes two dimensional and three dimensional Fourier descriptors are used to encode two and, respectively, three dimensional shapes. A pattern becomes apparent. Another direction of research would be to generalise the shared shape framework to other types of periodic functions with other types of Fourier descriptors. Example such functions, defined on spheres, are earthquake seismic maps. These functions can be encoded using the spherical Fourier descriptor. It would therefore be possible to apply the same framework to earthquake modelling and prediction.

Inference and training with GP-LVM becomes very slow as the number of data points in the training set increases beyond 3000 samples. This could be dealt with by using one of several sparsification techniques. Alternatively, we could learn local spaces. For example, for a space capturing the shapes of a person rotating around a single axis 180 degrees, we could use a separate space for each 30 degrees (so 6 local spaces in total). A way of switching between the various local spaces could be based on Bayesian model selection.

In a longer term, overall, the purpose of this research is to represent knowledge in a more human-like way. A longer term goal would therefore be the extension of the shared shape formulation presented in this work to involve hierarchical spaces that are learned automatically and online. The human mind represents concepts as hierarchies of details. Consider for example the concept of a face. At the lowest level all faces are seen as round blobs. One level up, a face is a round blob with three other round blobs inside (eyes and mouth). The higher up one goes in the hierarchy, the more details are added to the description. It is this very hierarchy that allows the human mind to cope with very large amounts of information, with huge in class and inter class variance and very high levels of noise. Currently, such a hierarchical

system is either not used, or domain specific and imposed artificial by the creator of the knowledge repository. Using this research it would be possible to learn these repositories automatically and independently of human intervention.

Appendix

A.1 fastHOG

The histogram of oriented gradients descriptor is one of the best and most popular descriptors in the computer vision literature. Unfortunately this technique suffers from one big problem: speed. Like most sliding window algorithms, it is very slow, making it unsuitable for any real time application.

In this appendix we detail an implementation of the HOG based sliding window algorithm using the NVIDIA CUDA framework. We achieve improvements of over 67x from the standard sequential implementation of Dalal and Triggs (2005). We are not the first to attempt such an implementation; notably Wojek et al. (2008). We believe our implementation to be closer to Dalal and Triggs (2005), and, as far as we can ascertain, our implementation is faster than Wojek et al. (2008). While Wojek et al. (2008) use the cheaper and more easily implemented bilinear interpolation, we mirror Dalal and Triggs (2005) in using trilinear; this translates into a significantly different implementation.

A.1.1 HOG Descriptor

The histogram of oriented gradients algorithm for object detection was introduced in Dalal and Triggs (2005). Example detection results for pedestrians and heads are shown in Figure A.1.1.

The HOG detector algorithm is a sliding window algorithm. This means that for any given image a window is moved across at all locations and scales and a descriptor is computed. For that window, a pretrained classifier is used to assign a matching score to the descriptor. The linear SVM classifier is used with a descriptor based on



Figure A.1.1: Example results after applying the HOG detector for pedestrians (left) and heads (right)

histograms of gradient orientations.

First, a gamma normalisation is applied on the image. The gamma normalisation has been proven to improve performance for pedestrian detection (Dalal and Triggs, 2005) but it may decrease performance for other object classes. To compute the gamma correction, the colour for each channel is replaced by its square root.

Gradient orientations and magnitudes are next obtained for each pixel from the preprocessed image. This is done by convoluting the image with the 1-D centred kernel [-1 0 1] by rows and columns. If a colour image is used the gradient with the maximum magnitude (and its corresponding orientation) is chosen.

Each detection window is divided into several groups of pixels, called cells. For each cell, a histogram of gradient orientations is computed. For pedestrian detection the cells are rectangular and the histogram bins are evenly spaced between 0 and 180 degrees. The contribution of each pixel to the cell histogram is weighted by a Gaussian centred in the middle of the block and then interpolated triliniarly in orientation and position. This has the effect of reducing aliasing.

Multiple cells are grouped into blocks, with each cell being part of multiple blocks. Each block therefore contains multiple cell histograms. Each block histogram is also normalised. For pedestrian detection the L2-Hys norm (Lowe, 2004) is used, followed by clipping to 0.2 and renormalising.

For pedestrian detection, the best results have been obtained using 16×16 pixel

blocks containing 2×2 cells of 8×8 pixels. The block stride is 8 pixels (one cell) so the histogram of a cell is normalised over 4 blocks. Each histogram has 9 bins. The Gaussian used for weighting pixel values has $\sigma = 8$. The detection window is 64×128. The scale ratio is 1.05.

A.1.2 Descriptor Computation

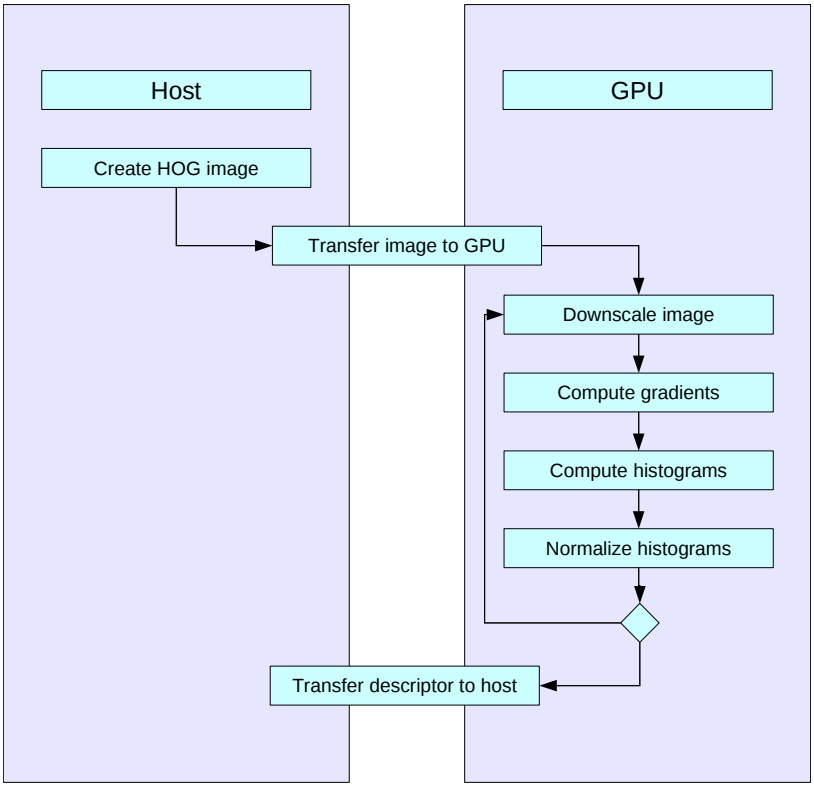


Figure A.1.2: fastHOG steps for computing the HOG descriptor

Our descriptor computation algorithm is split into 2 parts: the host and the GPU processing (as shown in Figure A.1.2). The image is acquired by the host, copied into the GPU memory, processed by the GPU and the descriptor is copied back to host memory. The following subsections detail our implementation.

Downscaling and Gamma Normalisation

We use the hardware texturing unit to downscale the image. This is different from the standard sequential implementation, where the downscale is implemented in software.

By copying the image inside a texture we can simply use the **tex2D** function to obtain a bilinearly interpolated value for each pixel. To minimise memory operations we also apply the gamma normalisation at this stage.

Colour Gradients

To compute the colour gradients we use two separable convolution kernels similar to the ones from the CUDA SDK. The first kernel convolutes the row with the centred 1-D mask, while the second kernel computes the column convolution, gradient orientations and magnitudes. Thread block sizes are fixed to 145 and 128 threads for row and column convolutions, respectively.

Our implementation supports two modes of operation: colour and grayscale. If colour is used, each thread computes three colour gradients, three magnitudes and three orientations. In the grayscale case only one gradient, magnitude and orientation need to be computed. Finally, for each pixel in the downscaled image two values are written to host memory: gradient orientation and magnitude. In the colour case the maximum magnitude (and corresponding orientation) is chosen.

Block Histograms

In the histogram computation step a HOG pixel block is mapped to a CUDA thread block. In the case of pedestrian detection, the block has 4 cells and each cell has 8 columns of 8 pixels, for a total of $8 \times 4 = 32$ threads.

We logically divide each thread block into 4 sub-blocks, each processing one cell. Each sub-block consists of 8 threads, each processing 8 pixels (one cell column). The thread configuration is shown in Figure A.1.3. While this thread model might not fully utilise the GPU hardware in some cases, it does have the advantage of scaling to different block/cell configurations.

In normal histogramming each pixel would be counted in its nearest orientation bin. This causes an aliasing effect, which may lead to sudden changes in the computed feature vector (Dalal and Triggs, 2005). To avoid this, trilinear linear interpolation of the pixel weights into the orientation histograms is used.

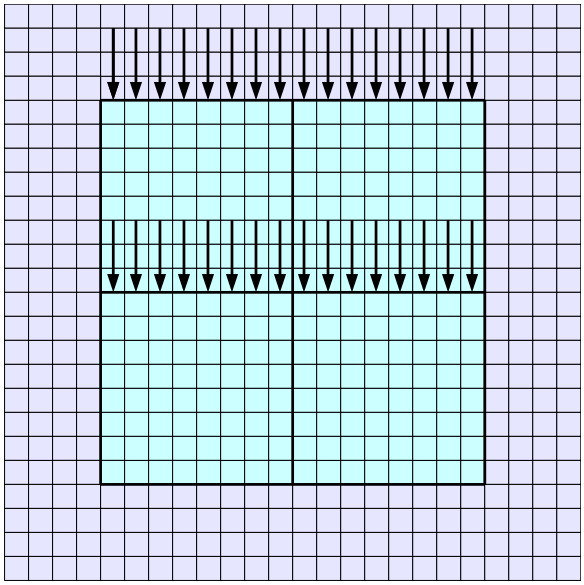


Figure A.1.3: Thread configuration for the block histogram step. The dark area represents one block, split into cells. The vertical arrows represent threads and the direction they process the pixels. Each thread processes one column from once cell in a block.

1	1 and 3	1 and 3	3
1 and 2	1, 2, 3 and 4	1, 2, 3, and 4	3 and 4
1 and 2	1, 2, 3 and 4	1, 2, 3 and 4	3 and 4
2	2 and 4	2 and 4	4

Figure A.1.4: A 2×2 block and the memory write pattern used for trilinear interpolation of the histogram bins. Each cell is split into 4 subcells. The pixels in the subcell marked with 1 contribute to the histogram of cell 1, the pixels in the subcell marked with 1 and 3 contribute to the histogram of cells 1 and 3, etc.

In our implementation each thread computes its own histogram and stores it in shared memory. Because of the trilinear interpolation the weight of a pixel is distributed into as many as 8 bins, from 4 different histograms. Each pixel will therefore contribute not only to its cell histogram but to one or more of the neighbouring ones, as depicted in Figure A.1.4. Because of this memory write pattern each thread must hold the complete block histogram (36 float values) in shared memory. This imposes certain restrictions on the block size and layout. Furthermore this means that it is not advantageous to keep the histogram weights in texture memory, as the memory read operations will be more costly than the actual computation of the weights.

In contrast, Wojek et al. (2008) use the bilinear interpolation technique, where each pixel contributes to fewer bins, so less shared memory is required and therefore a different thread block layout can be used. The weights do not need to be computed as it is more advantageous to read them from a texture. We chose to use trilinear interpolation in order to keep our implementation as close to the original one as possible and because trilinear histograms improve the performance of the detector (Dalal and Triggs, 2005).

Finally, the per thread histograms are merged using a parallel reduction algorithm. The Gaussian weights are stored in a texture and the values are written back to global memory in a 2D 18×2 grid. The parallel reduction is similar to the one from the **reduction** example from the SDK.

Histogram Normalisation

In the histogram normalisation step a HOG pixel block is mapped to a CUDA thread block. For the pedestrian detection case, each block consists of 4 histograms, each histogram having 9 bins. In our implementation each thread processes one pixel, so we use $9 \times 4 = 36$ threads.

Each block is kept in shared memory for the duration of the processing and wrote back to global memory in the same grid layout as in the previous step.

We use the parallel reduction once to compute the norm for each block, normalise each pixel value once, cap the value to 0.2, use the parallel reduction algorithm again and finally renormalise.

A.1.3 Linear SVM Evaluation

Often the descriptor needs to be used in some sort of classification. Dalal and Triggs (2005) for example use a linear SVM classifier to detect pedestrians. We also implemented this step on the GPU.

Each detection window is mapped to a CUDA thread block. For the pedestrian detection phase, assuming a detection window of 64×128 pixels, we use a 7×15

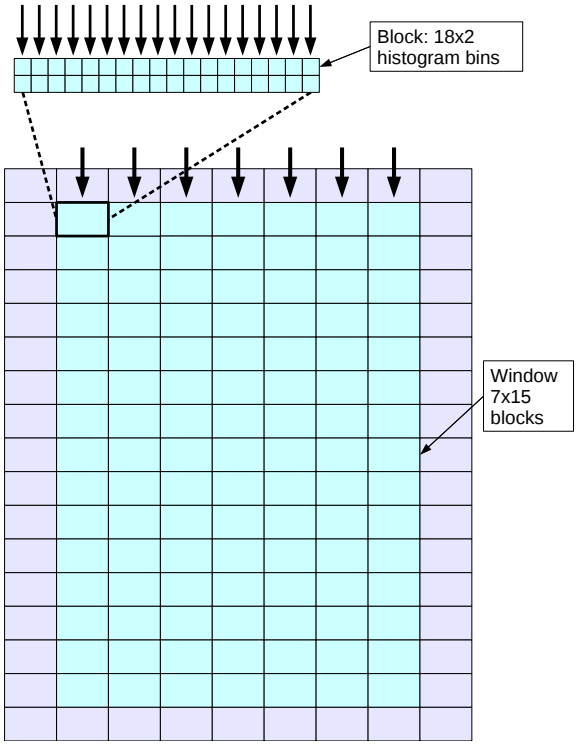


Figure A.1.5: Thread configuration for the SVM evaluation step. The vertical arrows represent threads, and the direction they process the pixels. For our case, where a window has 64×128 pixels equal to 7×15 blocks, a thread block has 7×18 threads, each thread processing 15×2 values.

CUDA thread block. Considering the grid layout used in the descriptor computation stage, each detection window is stored in global memory in a grid of 7×18 width and 15×2 height. We therefore have, for each block/detection window, 7×18 threads, each processing 15×2 values. The thread configuration is depicted in Figure A.1.5.

The value of each histogram bin is multiplied with its SVM weight, parallel reduction is used to add the weighted values and finally the hyperplane bias is subtracted. The final score is written back to global memory.

A.1.4 Benchmarks

Our implementation is designed to produce the same results as the normal CPU implementation of Dalal and Triggs (2005). Using our image resize method, the results produced by our implementation and the standard sequential one of Dalal

Implementation	Processing time
Ours (grayscale)	22s
Ours (colour)	31s
Wojek et al. (2008) GPU	1m 9s
Wojek et al. (2008) CPU	35m 1s

Table A.1.1: Processing time for the INRIA Person test set

and Triggs (2005) are identical. Therefore we believe that the tests done by Dalal and Triggs (2005) are valid for our implementation as well.

We tested our algorithm on a variety of images and video cards. Using a single GeForce GTX 285 video card we noticed about a 67x improvement over the normal sequential implementation. The processing time for one 640×480 image lies between *74ms* and *84ms*. We chose to keep the parameters (block size, etc.) flexible at the expense of some speed.

We ran our implementation on the INRIA dataset (Dalal and Triggs, 2005), testing our implementation with all 288 positive images. In Table A.1.1 we compare the processing time for our implementation (both colour and grayscale modes) with the Wojek et al. (2008) GPU and CPU implementations. When comparing GPU implementations, ours is about twice as fast in colour mode and about three times as fast in grayscale mode. We achieve a 67x speedup in colour mode and a 95x speedup in grayscale mode, over their CPU implementation.

Bibliography

Agarwal A, Triggs B (2004) 3D human pose from silhouettes by relevance vector regression. International Conference on Computer Vision and Pattern Recognition

Agarwal A, Triggs B (2006) Recovering 3D human pose from monocular images. IEEE Transactions on Pattern Analysis and Machine Intelligence 28:44–58

Armstrong M, Zisserman A (1995) Robust object tracking. In: Asian Conference on Computer Vision, pp 58–61

Athitsos V, Sclaroff S (2003) Estimating 3D hand pose from a cluttered image. In: International Conference on Computer Vision and Pattern Recognition, pp 432–439

Beis J, Lowe D (1997) Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: International Conference on Computer Vision and Pattern Recognition, pp 1000–1006

Bibby C, Reid I (2008) Robust real-time visual tracking using pixel-wise posteriors. In: European Conference on Computer Vision, pp 831–844

Binford T (1981) Inferring surfaces from images. Artificial Intelligence 17(1-3):205–244

Blake A, Curwen R, Zisserman A (1993) A framework for spatiotemporal control in the tracking of visual contours. International Journal of Computer Vision 11(2):127–145

Bosch A, Zisserman A, Munoz X (2007) Representing shape with a spatial pyramid kernel. In: International Conference on Image and Video Retrieval, pp 401–408

Bouguet JY (2008) Camera calibration toolbox for Matlab

Boykov Y, Kolmogorov V (2004) An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence pp 1124–1137

Brady M, Kersten D (2003) Bootstrapped learning of novel objects. Journal of Vision 3:413–422

Bray M, Koller-Meier E, Van Gool L (2004) Smart particle filtering for 3D hand tracking. In: International Conference on Automatic Face and Gesture Recognition, pp 675–680

Brox T, Rosenhahn B, Gall J, Cremers D (2009) Combined region- and motion-based 3D tracking of rigid and articulated objects. IEEE Transactions Pattern Analysis and Machine Intelligence 32(3):402–415

Chakraborty A, Staib L, Duncan J (1994) An integrated approach to boundary finding in medical images. In: IEEE Workshop Biomedical Image Analysis, pp 13–22

Comport A, Marchand E, Chaumette F (2003) A real-time tracker for markerless augmented reality. In: International Symposium on Mixed and Augmented Reality, pp 36–44

Cootes T, Taylor CJ, Cooper D, Graham J (1995) Active shape models – their training and application. Computer Vision and Image Understanding 61:38–59

Crammer K, Singer Y (2001) On the algorithmic implementation of multi-class SVMs. Journal of Machine Learning Research 2:265–292

Cremers D, Funka-lea G (2006) Dynamical statistical shape priors for level set based tracking. IEEE Transactions Pattern Analysis and Machine Intelligence 28:1262–1273

Cremers D, Osher S, Soatto S (2004) Kernel density estimation and intrinsic alignment for knowledge-driven segmentation: Teaching level sets to walk. In: International Journal of Computer Vision, pp 36–44

Cremers D, Rousson M, Deriche R (2007) A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape. International Journal of Computer Vision 72(2):195–215

Curwen R, Blake A (1993) Active vision

Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. International Conference on Computer Vision and Pattern Recognition pp 886–893

Dambreville S, Rathi Y, Tannenbaum A (2008a) A framework for image segmentation using shape models and kernel space shape priors. IEEE Transactions Pattern Analysis and Machine Intelligence 30(8):1385–1399

Dambreville S, Sandhu R, Yezzi A, Tannenbaum A (2008b) Robust 3D pose estimation and efficient 2D region-based segmentation from a 3D shape prior. In: European Conference on Computer Vision, pp 169–182

de Campos T, Murray D (2006) Regression-based hand pose estimation from multiple cameras. In: International Conference on Computer Vision and Pattern Recognition, pp 782–789

de La Gorce M, Paragios N, Fleet D (2008) Model-based hand tracking with texture, shading and self-occlusions. In: International Conference on Computer Vision and Pattern Recognition, pp 1–8

Doucet A, De Freitas N, Gordon N (eds) (2001) Sequential Monte Carlo methods in practice

Drummond T, Cipolla R (2002) Real-time visual tracking of complex structures. IEEE Transactions Pattern Analysis and Machine Intelligence 24:932–946

Ek CH (2009) Shared gaussian process latent variable models. PhD thesis, Oxford Brookes University

Ek CH, Torr P, Lawrence N (2007) Gaussian process latent variable models for human pose estimation. In: Joint Workshop on Machine Learning and Multimodal Interaction, pp 132–143

Ek CH, Rihan J, Torr P, Rogez G, Lawrence N (2008) Ambiguity modeling in latent spaces. In: Joint Workshop on Machine Learning and Multimodal Interaction, pp 62–73

Everingham M, van Gool L, Williams C, Winn J, Zisserman A (2010) The pascal visual object classes (voc) challenge. International Journal of Computer Vision 88(2):303–338

Fang CY, Chen SW, Fuh CS (2003) Road-sign detection and tracking. Transactions on Vehicular Technology 52:1329–1341

Farahani S (2008) ZigBee Wireless Networks and Transceivers

Felzenszwalb P, Huttenlocher D (2004) Distance transforms of sampled functions. Tech. rep., Cornell Computing and Information Science

Fischler M, Bolles R (1987) Readings in computer vision: issues, problems, principles, and paradigms. chap Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, pp 726–740

Freeman W (1993) Exploiting the generic view assumption to estimate scene parameters. In: International Conference on Computer Vision, pp 347–356

Freescale (2007) Tilt sensing using linear accelerometers, AN3461

Gall J, Rosenhahn B, Seidel HP (2008) Drift-free tracking of rigid and articulated objects. International Conference on Computer Vision and Pattern Recognition pp 1–8

Gilbert J, Nocedal J (1992) Global convergence properties of conjugate gradient methods for optimization. SIAM Journal on Optimization 2(1):21–42

Grenander U, Chow Y, Keenan D (1991) Hands: A Pattern Theoretic Study of Biological Shapes

Hager W, Zhang H (2005) A new conjugate gradient method with guaranteed descent and an efficient line search. SIAM Journal on Optimization 16(1):170–192

Harris C (1993) Tracking with rigid models. Active Vision pp 59–73

Hertzmann A (2006) Learning human pose and motion models for animation

Hol J, Schön T, Gustafsson F (2010) Modelling and calibration of inertial and vision sensors. International Journal of Robotics Research 29:231–244

Isard M, Blake A (1998) CONDENSATION - conditional density propagation for visual tracking. International Journal of Computer Vision 29:5–28

Kass M, Witkin A, Terzopoulos D (1988) Snakes: Active contour models. International Journal of Computer Vision 1(4):321–331

Kersten D, Mamassian P, Yuille A (2004) Object Perception as Bayesian Inference.

Annual Review of Psychology 55(1):271–304

Khayam SA (2003) The Discrete Cosine Transform (DCT): Theory and Application

Klein G, Murray D (2006) Full-3D Edge Tracking with a Particle Filter

Klein G, Murray D (2007) Parallel tracking and mapping for small AR workspaces. In: International Symposium on Mixed and Augmented Reality, pp 1–10

Koller D, Daniilidis K, Nagel HH (1993) Model-based object tracking in monocular image sequences of road traffic scenes. International Journal of Computer Vision 10:257–281

Kollnig H, Nagel HH (1997) 3d pose estimation by directly matching polyhedral models to gray value gradients. International Journal of Computer Vision 23:283–302

Kourtzi Z, Kanwisher N (2001) Representation of perceived object shape by the human Lateral Occipital Complex. Science 293:1506–1509

Kuhl F, Giardina C (1982) Elliptic fourier features of a closed contour. Computer Graphics and Image Processing 18(3):236–258

Kwok J, Tsang I (2003) The pre-image problem in kernel methods. In: International Conference on Machine Learning, pp 408–415

Lang P, Pinz A (2005) Calibration of hybrid vision / inertial tracking systems. In: Workshop on Integration of Vision and Inertial Systems

Lawrence N (2005) Probabilistic non-linear principal component analysis with gaussian process latent variable models. Journal of Machine Learning Research 6:1783–1816

Lawrence N, Quinonero-Candela J (2006) Local distance preservation in the gp-lvm through back constraints. In: International Conference on Machine Learning, pp 513–520

de Leeuw J (2005) Modern multidimensional scaling: Theory and applications (second edition). Journal of Statistical Software, Book Reviews 14(4):1–2

Lepetit V, Fua P (2005) Monocular model-based 3D tracking of rigid objects: A survey. Foundations and Trends in Computer Graphics and Vision 1(1):1–89

Lepetit V, Lagger P, Fua P (2005) Randomized trees for real-time keypoint recognition. In: International Conference on Computer Vision and Pattern Recognition, pp 775–781

Leventon M, Grimson E, Faugeras O (2000) Statistical shape influence in geodesic active contours. In: International Conference on Computer Vision and Pattern Recognition, vol 1, pp 316–323 vol.1

Lin D, Wu Y, Huang TST (2002) Capturing human hand motion in image sequences. In: Workshop on Motion and Video Computing, pp 99–104

Lopez LD, Fuentes O (2007) Color-based road sign detection and tracking. In: International Conference on Image Analysis and Recognition, pp 1138–1147

Lowe D (1992) Robust model-based motion tracking through the integration of search and estimation. International Journal of Computer Vision 8(2):113–122

Lowe D (2004) Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2):91–110

Lu C, Hager G, Mjolsness E (2000) Fast and globally convergent pose estimation from video images. IEEE Transactions Pattern Analysis and Machine Intelligence 22:610–622

Lu S, Metaxas D, Samaras D (2003) Using multiple cues for hand tracking and model refinement. In: International Conference on Pattern Recognition, pp 443–450

MacCormick J, Isard M (2000) Partitioned sampling, articulated objects, and interface-quality hand tracking. In: European Conference on Computer Vision, pp 3–19

Maes P (2009) SixthSense: Integrating information and the real world. In: IEEE International Symposium on Mixed and Augmented Reality, pp 21–27

Mahalanobis PC (1936) On the generalised distance in statistics. In: Proceedings National Institute of Science, India, vol 2, pp 49–55

Marchand E, Bouthemy P (2001) A 2d-3d model-based approach to real-time visual tracking. Image and Vision Computing 19:941–955

Marr D, Hildreth E (1980) Theory of Edge Detection. Proceedings of the Royal Society of London Series B Biological Sciences 207(1167):187–217

Measurand Inc (2009) Shapehand data glove. URL http://www.shapehand.com/

Menet S, Saint-Marc P, Medioni G (1990) B-Snakes:implementation and application to stereo. In: Image Understanding Workshop, pp 720–726

Meuter M, Kummert A, Muller-Schneiders S (2008) 3d traffic sign tracking using a particle filter. In: Intelligent Transportation Systems, pp 168–173

Mika S, Schölkopf B, Smola A, Scholz M, Rätsch G (1999) Kernel PCA and de-noising in feature spaces. In: Neural Information Processing Systems, pp 536–542

Moeslund T, Hilton A, Krüger V (2006) A survey of advances in vision-based human motion capture and analysis. Computer Vision and Image Understanding 104(2-3):90–126

Moré J (1978) The Levenberg-Marquardt algorithm: Implementation and theory. In: Numerical Analysis, vol 630, chap 10, pp 105–116

Navaratnam R, Fitzgibbon A, Cipolla R (2007) The joint manifold model for semi-supervised multi-valued regression. In: International Conference on Computer Vision and Pattern Recognition, pp 1–8

NVIDIA (2009) NVIDIA CUDA Programming Guide 2.2

Osher S, Sethian J (1988) Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. Journal of Computers 79(1):12–49

Ozuysal M, Lepetit V, Fleuret F, Fua P (2006) Feature harvesting for tracking-by-detection. In: European Conference on Computer Vision, pp 592–605

Ozuysal M, Fua P, Lepetit V (2007) Fast keypoint recognition in ten lines of code. In: International Conference on Computer Vision and Pattern Recognition, pp 1–8

Pavlovic V, Sharma R, Huang T (1997) Visual interpretation of hand gestures for human-computer interaction: A review. IEEE Transactions Pattern Analysis and Machine Intelligence 19:677–695

Piccioli G, de Micheli E, Campani M (1994) A robust method for road sign detection and recognition. In: European Conference on Computer Vision, pp 495–500

Press W (2007) Numerical Recipes: the Art of Scientific Computing, 3rd edn. Cambridge University Press

Prisacariu VA, Reid I (2009a) fastHOG - a real-time GPU implementation of HOG. Tech. Rep. 2310/09, Department of Engineering Science, Oxford University

Prisacariu VA, Reid I (2009b) PWP3D: Real-time segmentation and tracking of 3D objects. In: British Machine Vision Conference, pp 1–10

Prisacariu VA, Reid I (2011a) Nonlinear shape manifolds as shape priors in level set segmentation and tracking. In: International Conference on Computer Vision and Pattern Recognition, pp 2185–2192

Prisacariu VA, Reid I (2011b) Robust 3D hand tracking for human computer interaction. In: International Conference on Automatic Face and Gesture Recognition, pp 368–375

Prisacariu VA, Reid I (2011c) Shared shape spaces. In: International Conference on Computer Vision

Prisacariu VA, Reid I (2012a) 3D Hand Tracking for Human Computer Interaction. Image and Vision Computing (0)

Prisacariu VA, Reid I (2012b) PWP3D: Real-Time Segmentation and Tracking of 3D Objects. International Journal of Computer Vision pp 1–20

Prisacariu VA, Timofte R, Zimmermann K, Reid I, Van Gool L (2010) Integrating object detection with 3D tracking towards a better driver assistance system. In: International Conference on Pattern Recognition, pp 3344–3347

Rasmussen C, Williams C (2005) Gaussian Processes for Machine Learning

Rathi Y, Dambreville S, Tannenbaum A (2006) Statistical shape analysis using kernel PCA. In: International Society for Optical Engineering, vol 6064, pp 425–432

Ravela S, Draper B, Lim J, Weiss R (1995) Adaptive tracking and model registration across distinct aspects. Intelligent Robots and Systems pp 174–180

Rosenhahn B, Brox T, Weickert J (2007) Three-dimensional shape knowledge for joint image segmentation and pose tracking. International Journal of Computer Vision 73(3):243–262

Rother C, Kolmogorov V, Blake A (2004) Grabcut: interactive foreground extraction

using iterated graph cuts. ACM Transactions on Graphics 23(3):309–314

Rousson M, Paragios N (2002) Shape priors for level set representations. In: European Conference on Computer Vision, pp 78–92

Ruta A, Li Y, Liu X (2010) Real-time traffic sign recognition from video by class-specific discriminative features. Pattern Recognition 43:416–430

Sandhu R, Dambreville S, Yezzi A, Tannenbaum A (2009) Non-rigid 2D-3D pose estimation and 2D image segmentation. In: International Conference on Computer Vision and Pattern Recognition, pp 786–793

Sandhu R, Dambreville S, Yezzi A, Tannenbaum A (2011) A Nonrigid Kernel-Based Framework for 2D-3D Pose Estimation and 2D Image Segmentation. IEEE Transactions Pattern Analysis and Machine Intelligence 33(6):1098–1115

Scharr H (2000) Optimal operators in digital image processing. PhD thesis

Schmaltz C, Rosenhahn B, Brox T, Cremers D, Weickert J, Wietzke L, Sommer G (2007a) Region-based pose tracking. In: Iberian Conference on Pattern Recognition and Image Analysis, pp 56–63

Schmaltz C, Rosenhahn B, Brox T, Weickert J, Cremers D, Wietzke L, Sommer G (2007b) Occlusion modelling by tracking multiple objects. In: German Association for Pattern Recognition DAGM Conference, pp 173–183

Schölkopf B, Smola A, Müller KR (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10:1299–1319

Schweighofer G, Pinz A (2006) Robust pose estimation from a planar target. IEEE Transactions Pattern Analysis and Machine Intelligence 28:2024–2030

Shen C, Lin X, Shi Y (2009) Human pose estimation from corrupted silhouettes using a sub-manifold voting strategy in latent variable space. Pattern Recognition Letters 30(4):421–431

Sigal L, Memisevic R, Fleet D (2009) Shared kernel information embedding for discriminative inference. International Conference on Computer Vision and Pattern Recognition

Sigal L, Balan A, Black M (2010) HumanEva: Synchronized Video and Motion Capture Dataset and Baseline Algorithm for Evaluation of Articulated Human Motion. International Journal of Computer Vision 87:4–27

Simon G, Berger MO (1998) A two-stage robust statistical method for temporal registration from features of various type. In: International Conference on Computer Vision, pp 261–267

Skrypnyk I, Lowe D (2004) Scene modelling, recognition and tracking with invariant image features. In: International Symposium on Mixed and Augmented Reality 2004, pp 110–119

Staib L, Duncan J (1992) Boundary Finding with Parametrically Deformable Models. IEEE Transactions Pattern Analysis and Machine Intelligence 14(11):1061–1075

Stenger B, Thayananthan A, Torr P, Cipolla R (2003) Filtering using a tree-based estimator. In: International Conference on Computer Vision, vol 2, pp 1063–1071

Stenger B, Thayananthan A, Torr P, Cipolla R (2006) Model-based hand tracking using a hierarchical bayesian filter. IEEE Transactions on Pattern Analysis and Machine Intelligence 28:1372–1384

Timofte R, Zimmermann K, Van Gool L (2009) Multi-view traffic sign detection, recognition, and 3d localisation. In: Workshop on the Applications of Computer Vision

Timofte R, Prisacariu VA, Van Gool L, Reid I (2011) Chapter 3.5: Combining Traffic Sign Detection with 3D Tracking Towards Better Driver Assistance. In: Emerging Topics in Computer Vision and its Applications, World Scientific Publishing

TinEye (2010) Tineye commercial api http://www.tineye.com

Tipping M, Bishop C (1999) Probabilistic principal component analysis. Journal of the Royal Statistical Society, Series B 61:611–622

Tsai A, Yezzi A, Wells W, Tempany C, Tucker D, Fan A, Grimson E, Willsky A (2003) A shape-based approach to the segmentation of medical imagery using level sets. Transactions on Medical Imaging 22(2):137–154

Uenohara M, Kanade T (1991) Vision-based object registration for real-time image overlay. In: Journal of Cognitive Neuroscience, pp 13–22

Urtasun R, Darrell T (2008) Sparse probabilistic regression for activity-independent human pose inference. In: International Conference on Computer Vision and Pattern Recognition, pp 1–8

Urtasun R, Fleet D, Hertzmann A, Fua P (2005) Priors for people tracking from small training sets. In: International Conference on Computer Vision, vol 1, pp 403–410

Urtasun R, Fleet D, Fua P (2006) 3d people tracking with gaussian process dynamical models. In: International Conference on Computer Vision and Pattern Recognition, pp 238–245

UZR GmbH (2009) iModeller 3D Professional. URL http://www.imodeller.com/

Vacchetti L, Lepetit V, Fua P (2004a) Combining edge and texture information for real-time accurate 3d camera tracking. In: International Symposium on Mixed and Augmented Reality, pp 48–57

Vacchetti L, Lepetit V, Fua P (2004b) Stable real-time 3d tracking using online and offline information. IEEE Transactions Pattern Analysis and Machine Intelligence 26:1385–1391

Vallance W, Hodge D, Pedoe D (1952) Methods of algebraic geometry. 3

Vese L, Chan T (2002) A multiphase level set framework for image segmentation using the mumford and shah model. International Journal of Computer Vision 50(3):271–293

Viola P, Jones M (2001) Robust real-time face detection. In: International Conference

on Computer Vision, vol 2, pp 747–757

Wagner D, Reitmayr G, Mulloni A, Drummond T, Schmalstieg D (2008) Pose tracking from natural features on mobile phones. In: International Symposium on Mixed and Augmented Reality, pp 125–134

Wang R, Popović J (2009) Real-time hand-tracking with a color glove. ACM Transactions on Graphics 28

Wang Y, Staib L (1998) Boundary finding with correspondence using statistical shape models. In: International Conference on Computer Vision and Pattern Recognition, pp 338–346

Wojek C, Dorkó G, Schulz A, Schiele B (2008) Sliding-windows for rapid object class localization: A parallel technique. In: German Association for Pattern Recognition DAGM Conference, pp 71–81

Yezzi A, Tsai A, Willsky A (1999) A statistical approach to snakes for bimodal and trimodal imagery. In: International Conference on Computer Vision, pp 898–906

Yuille A, Hallinan P, Cohen D (1992) Feature extraction from faces using deformable templates. International Journal of Computer Vision 8(2):99–111