



University
of Glasgow

Ponder, Christopher John (2015) A generic computer platform for efficient iris recognition. EngD thesis.

<http://theses.gla.ac.uk/6780/>

Copyright and moral rights for this thesis are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the Author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the Author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

A Generic Computer Platform For Efficient Iris Recognition

Christopher John Ponder BEng (Hons)

A thesis submitted to the Universities of

Edinburgh,
Glasgow,
Heriot-Watt,
and Strathclyde

For the Degree of Doctor of Engineering
in System Level Integration

© Christopher John Ponder 2014

To Rachel, Alex and Ben

Abstract

This document presents the work carried out for the purposes of completing the Engineering Doctorate (EngD) program at the Institute for System Level Integration (iSLI), which was a partnership between the universities of Edinburgh, Glasgow, Heriot-Watt and Strathclyde. The EngD is normally undertaken with an industrial sponsor, but due to a set of unforeseen circumstances this was not the case for this work. However, the work was still undertaken to the same standards as would be expected by an industrial sponsor.

An individual's biometrics include fingerprints, palm-prints, retinal, iris and speech patterns. Even the way people move and sign their name has been shown to be uniquely associated with that individual. This work focuses on the recognition of an individual's iris patterns.

The results reported in the literature are often presented in such a manner that direct comparison between methods is difficult. There is also minimal code resource and no tool available to help simplify the process of developing iris recognition algorithms, so individual developers are required to write the necessary software almost every time. Finally, segmentation performance is currently only measurable using manual evaluation, which is time consuming and prone to human error.

This thesis presents a completely novel generic platform for the purposes of developing, testing and evaluating iris recognition algorithms which is designed to simplify the process of developing and testing iris recognition algorithms. Existing open-source algorithms are integrated into the generic platform and are evaluated using the results it produces.

Three iris recognition segmentation algorithms and one normalisation algorithm are proposed. Three of the algorithms increased true match recognition performance by between two and 45 percentage points when compared to the available open-source algorithms and methods found in the literature. A matching algorithm was developed that significantly speeds up the process of analysing the results of encoding. Lastly, this work also proposes a method of automatically evaluating the performance of segmentation algorithms, so minimising the need for manual evaluation.

Declaration of Originality

I, Christopher John Ponder, declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

The material contained in this thesis is my own original work produced under the supervision of Khaled Benkrid and Martin Reekie.

Dedication and Acknowledgements

This thesis is dedicated to my family, Rachel, Alex & Ben, whose love, encouragement and understanding made this possible.

I would like to thank Khaled Benkrid, my academic supervisor for the first three years of my research, for believing in me and for his support and encouragement. I would like to express my sincere gratitude and deepest appreciation to Martin Reekie who has been there for me throughout my journey and has been my academic supervisor for the last two years. Their support and encouragement over the course of my studies have been invaluable.

I would like to thank Theresa, for her understanding, kind words and sympathetic ear, without which this would have been a far more difficult journey.

I would also like to thank the people at Tritech International Limited for their patience, understanding and timely assistance during the writing of this thesis.

Finally, I would like to thank the all staff at the iSLI and Glasgow University for their help and support throughout my studies.

Portions of the research in this thesis use the CASIA-Iris V1, V2, V3 and V4 image datasets collected by the Centre for Biometrics and Security Research at the Chinese Academy of Sciences Institute of Automation, <http://biometrics.idealtest.org/>, and <http://www.cbsr.ia.ac.cn/IrisDataset.htm>

Portions of the research in this thesis use the MMU1 and MMU2 image datasets, <http://pesona.mmu.edu.my/~ccteo/>

Portions of the research in this thesis use the UBIRIS V1 and V2 image datasets collected by Hugo Pedro Proença and Luís A. Alexandre of Department of Computer Science, University of Beira Interior, <http://iris.di.ubi.pt/ubiris1.html> and <http://iris.di.ubi.pt/ubiris2.html>

Portions of the research in this thesis were tested using version 1.0 of the IITD Iris Database http://www4.comp.polyu.edu.hk/~csajaykr/IITD/Database_Iris.htm

Table of Contents

Abstract	3
Dedication and Acknowledgements	5
List of Figures	12
List of Tables	19
List of Image Datasets	23
List of Symbols	24
Glossary	27
1 Introduction	34
1.1 Motivation.....	35
1.2 Thesis Outline	39
1.3 Thesis Outcomes.....	43
2 Literature Review	45
2.1 Image Capture.....	48
2.2 Pre-Processing	54
2.3 Segmentation	56
2.4 Normalisation.....	61
2.5 Encoding	64
2.6 Matching	68
2.7 Generic Platform.....	75
2.8 Conclusions.....	77
3 Designing the Generic Platform	78
3.1 Development Environment and Coding Language.....	81
3.1.1 Background and Thinning the Field.....	81
3.1.2 C/C++	82
3.1.3 Java.....	83
3.1.4 MATLAB	84
3.1.5 FreeMAT	85
3.1.6 Octave.....	86
3.1.7 SciLab.....	86

3.1.8	Conclusion.....	87
3.2	Measurable Algorithm Performance via Results.....	90
3.2.1	Genuine versus Imposter Hamming Distance Graph.....	90
3.2.2	Decidability.....	94
3.2.3	FRR, FAR, EER and ROC.....	95
3.2.4	Conclusion.....	98
3.3	Changing Iris Operations without Recompilation.....	101
3.4	In-built Iris Recognition Operations.....	103
3.5	Fast Iris Recognition Algorithm Swapping.....	104
3.6	Image Datasets.....	105
3.7	Image Dataset Selection.....	108
3.7.1	CASIA V1 [4].....	108
3.7.2	CASIA V2 [5].....	109
3.7.3	CASIA V3 [6].....	109
3.7.4	CASIA V4 [7].....	111
3.7.5	IIT Delhi [13, 14].....	111
3.7.6	MMU [8].....	112
3.7.7	WVU [11, 12].....	112
3.7.8	UBIRIS [9] [10].....	113
3.7.9	UPOL [15, 16].....	114
3.7.10	ICE 2005 [17].....	114
3.7.11	Summary of Datasets.....	115
3.7.12	Selection of Datasets for Testing.....	119
3.8	The Generic GUI.....	123
3.9	Conclusions.....	137
4	Evaluating the Generic Platform.....	139
4.1	Developing the Generic Platform with Masek's Iris Recognition Algorithm.....	143
4.1.1	Results.....	145
4.1.2	Conclusions.....	153
4.2	Further Dataset Evaluation with Masek's Algorithm.....	154
4.2.1	Conclusions.....	170
4.3	Adding Other Open-source Iris Recognition Algorithms.....	172

4.3.1	Antonino's Encoding and Matching Code	173
4.3.2	Li's Iris Recognition Code	178
4.3.3	The JIRRM Segmentation and Normalisation Code.....	187
4.3.4	The OSIRIS Iris Recognition Code.....	194
4.3.5	Conclusions	199
4.4	Downloadable Iris Recognition Algorithms that were not Used.....	201
4.4.1	Iris Recognition System	202
4.4.2	High Performance Iris Recognition.....	204
4.4.3	GIRIST	207
4.5	Testing Segmentation Algorithms from the Literature Using the Generic Platform.....	209
4.5.1	Lin <i>et al.</i> 's Segmentation Algorithm.....	210
4.5.2	Lu's Segmentation Algorithm.....	211
4.5.3	The Shamsi-Lin Segmentation Algorithm	212
4.5.4	Results	213
4.5.5	Conclusions	217
4.6	Adding Support Functions	219
4.6.1	Edge Detection Routines.....	221
4.6.2	Hough Circle Detection Routine	224
4.6.3	Conclusions	226
4.7	Evaluating the Performance of MATLAB.....	227
4.7.1	Performance of the 'Fixed Flow' compared against the 'Custom Flow'	228
4.7.2	Determining the Performance Increase of Converting MATLAB code to OS Native code.....	229
4.7.3	The Performance Effects the OS and PC Hardware	230
4.7.4	Boosting Performance of the Generic Platform Using Pre-calculation	233
4.7.5	Conclusions	235
4.8	Chapter Conclusions	236
5	Algorithms Developed as Part of this Work	238
5.1	Segment One.....	241
5.1.1	Segment One Results	243
5.2	Segment Two	246
5.2.1	Initial Segment Two Results	253
5.2.2	Sum-of-Columns Method.....	256

5.2.3	Final Segment Two Results.....	260
5.2.4	Conclusions	261
5.3	Segment Three	262
5.3.1	Segment Three Results.....	266
5.3.2	Conclusions	269
5.4	Automatic Segmentation Evaluation	270
5.4.1	Automatic Segmentation Results	273
5.4.2	Conclusions	275
5.5	Normalise One	276
5.6	Improving Recognition Performance During Normalisation	280
5.6.1	Intelligent Noise Infilling of Noise Within the unwrapped iris	282
5.6.2	Noise Removal Using Standard Deviation Thresholding	285
5.6.3	Oversampling to Improve Noise Immunity	289
5.6.4	Normalise Two.....	291
5.6.5	Conclusions	294
5.7	Segmentation Mask One (SM1)	295
5.8	Match One.....	296
5.8.1	Results	303
5.8.2	Conclusions	305
5.9	Compare Result Databases	306
5.10	Comparison of the Methods Developed with the Literature.....	309
5.11	Chapter Conclusions	314
6	Conclusions and Future Work	316
6.1	Summary and Conclusions	316
6.2	Future work.....	322
Appendix A	Histogram Thresholding to Find the Pupil Boundary	325
Appendix B	Erroneously Good Results from the Similarity Between Images.....	328
Appendix C	The Hough Transform	330
Appendix D	The Circle Bisector Method.....	335
Appendix E	Published Paper, ICB 2012.....	337
Appendix F	Filenames of Images Removed from CASIA V1	345

Appendix G	Example Analysis Output	347
Appendix H	Quick-Start User Manual	350
H.1	Installation and Setup of the Generic Platform.....	350
H.2	Getting Started With Using the Generic Platform	358
7	Bibliography.....	362

List of Figures

Figure 2.1 – Image 6_3 from the WVU Free image dataset, with each of the features of interest for the purposes of iris recognition	45
Figure 2.2 – A block diagram of Daugman’s iris image-capture setup [51].....	49
Figure 2.3 – A block diagram of the iris image-capture setup used by Wildes <i>et al.</i> [51] ..	50
Figure 2.4 – Example of a close-up iris image capture device being used [52]	51
Figure 2.5 – CASIA V3i image S1004R05 with segmentation circles determined by Segment Three iris segmentation algorithm discussed in Section 5.3.....	56
Figure 2.6 – Edge image resulting from the integro-differential operator being applied to CASIA V3i image S1004R05	58
Figure 2.7 – Edge image resulting from the Canny edge detector being applied to CASIA V3i image S1004R05	59
Figure 2.8 – CASIA V3i image S1004R05 with segmentation circles.....	61
Figure 2.9 – Daugman’s rubber sheet model [77, 31], translating from Cartesian to polar coordinates	61
Figure 2.10 – Daugman’s rubber sheet model [77, 31] providing pupil displacement correction showing exaggerated pupil displacement for illustration	62
Figure 2.11 – Unwrapped iris pattern of Figure 2.8.....	62
Figure 2.12 – Mask for Figure 2.11 unwrapped iris pattern	63
Figure 2.13 – Daugman’s phase quantisation code sequence [76]	65
Figure 2.14 – Daugman iris code of Figure 2.11	65
Figure 2.15 – Wildes multi-scale iris code [51] showing the results of successive sub-sampling of the lower frequency bands of the iris pattern.....	66
Figure 2.16 – Example iris codes with XOR calculation and Hamming distance calculation	69
Figure 2.17 – Example iris codes with XOR calculation and Hamming distance calculation. Code 2 has been shifted once to the left.....	70
Figure 2.18 – Example iris codes with XOR calculation and Hamming distance calculation. Code 2 has been shifted left a second time.	70
Figure 2.19 – Example iris codes with XOR calculation, the AND operation using masks and the Hamming distance calculation	70
Figure 2.20 – Example iris codes with XOR calculation, the AND operation using masks and the Hamming distance calculation. Code 2 has been shifted once to the left	71

Figure 2.21 – Example iris codes with XOR calculation, the AND operation using masks and the Hamming distance calculation. Code 2 has been shifted left a second time.....	72
Figure 3.1 – A generic Daugman iris code generation process.....	79
Figure 3.2 – “Statistical Decision Theory: general formulation for decisions under uncertainty” [1]	91
Figure 3.3 – Example Hamming distance distribution for authentic and imposters produced using Segmentation Three on the CASIA V3-interval image dataset. Graph generated using the generic platform.	92
Figure 3.4 – Example Hamming distance distribution for authentic and imposters produced using Antonino’s algorithm on the MMU V1 image dataset. Graph generated using the generic platform.....	93
Figure 3.5 – Example FAR against FRR graph for Segment Three with Normalise Three, Masek’s encoded and Match Three using the CASIA V1 dataset within the generic platform.	95
Figure 3.6 – The ROC curve from Lin <i>et al.</i> 's method [30].....	96
Figure 3.7 – An almost ideal 1-FRR ROC graph produced using the generic platform with Segmentation Three using the IITD dataset.....	97
Figure 3.8 – Input image example from CASIA V1 database	103
Figure 3.9 – Binary biometric template, code (top) with mask (bottom)	103
Figure 3.10 – Iris GUI requesting the image database root directory	106
Figure 3.11 – Iris GUI image dataset list	106
Figure 3.12 – Generic Platform GUI running PC1 with ‘Manual Process’ selected and GUI regions numbered in blue	125
Figure 3.13 – Generic Platform GUI running PC1 with ‘Auto Process’ selected and GUI regions numbered in blue	126
Figure 3.14 – Flow chart depicting the selection choices available options for iris recognition algorithms added as a fixed flow.	127
Figure 3.15 – Generic Platform User Experience Flowchart.....	130
Figure 3.16 – Generic platform normalisation pane showing a normalised iris with its mask and two debug images.....	131
Figure 3.17 – Image (16) from Figure 3.16 when opened in a separate figure window....	133
Figure 3.18 – Generic platform encoding pane showing an iris code with its mask	133
Figure 3.19 – Generic platform matching pane showing the results from matching the currently selected image (10_3.bmp from WVU Free dataset)	134

Figure 3.20 – Generic platform results pane showing the reference segmentation for image 010204 from the MMU V2 image dataset.	135
Figure 3.21 – Area (24) in Figure 3.20 after selecting the ‘I’ button to zoom in to the iris boundary.....	136
Figure 3.22 – Area (24) in Figure 3.20 after selecting the ‘P’ button to zoom in to the pupil boundary.....	136
Figure 4.1 – FAR and FRR curves for Masek’s thesis results using CASIA-a image dataset (Left) and Masek’s algorithm (Right) when executed within the generic platform using CASIA V1 image dataset.....	148
Figure 4.2 – The normalised Hamming distribution for Masek’s flow using CASIA V1 produced using the generic platform.....	152
Figure 4.3 – Images from the CASIA V2D1 dataset that have been segmented by Masek’s algorithm.	156
Figure 4.4 – Image 001_1_1 from CASIA V1 showing segmentation masking by Masek’s algorithm (a) and with threshold applied (b) with no threshold applied.....	157
Figure 4.5 – Image 0000_10 from CASIA V2D1 showing segmentation masking by Masek’s algorithm (a) and with threshold applied (b) with no threshold applied.....	158
Figure 4.6 – unwrapped iris (a) and mask (b) of Figure 4.4a and unwrapped iris (c) and mask (d) of Figure 4.4b.....	159
Figure 4.7 – unwrapped iris (a) and mask (b) of Figure 4.5a and unwrapped iris (c) and mask (d) of Figure 4.5b.....	160
Figure 4.8 – True match scores for each image dataset when used with Masek’s algorithm using a range of threshold values.....	165
Figure 4.9 – Graph plotting the number of images in a dataset against the time taken to analyse the recognition performance in minutes using Masek’s matching algorithm within the generic platform.....	167
Figure 4.10 – Graph plotting the number of pixels in images in each dataset against the average time in seconds taken to create the iris code for images in the dataset using Masek’s matching algorithm within the generic platform.....	168
Figure 4.11 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino’s algorithm.....	175
Figure 4.12 – Comparison of analysis times from Masek’s algorithm with no thresholding (NT) against the analysis times attained by Antonino’s algorithm.....	177
Figure 4.13 – Images 1_0 (left) and 1_2 (right) from WVU Free image dataset, the yellow ellipses indicate the detected iris boundaries.....	179

Figure 4.14 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino and Li’s algorithms.	182
Figure 4.15 – Hamming distance distribution plots for Li’s segmentation algorithm using Normalise One and Masek’s encoding and matching routines.	185
Figure 4.16 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino and Li’s (Flow, S&N and S) algorithms.	186
Figure 4.17 – Comparison of true match results from JIRRM against the true match results attained by Antonino, Masek and Li’s algorithms.	189
Figure 4.18 – Comparison of code creation times with Masek’s algorithm against the times attained by JIRRM and Li’s algorithms.	190
Figure 4.19 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino, JIRRM, JIRRM with more circles (JIRRM MC) and Li’s algorithms.	192
Figure 4.20 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino’s encoding and matching algorithm, and JIRRM, OSIRIS and Li’s segmentation algorithms.	198
Figure 4.21 – Comparison of true match results from Masek, Antonino, JIRRM, OSIRIS and Li’s algorithms.	199
Figure 4.22 – Iris Recognition with CASIA V1 image 001_1_1.bmp loaded.	202
Figure 4.23 – High Performance Iris Recognition GUI.	205
Figure 4.24 – GIRIST GUI with a library of CASIA V1 loaded.	207
Figure 4.25 – GIRIST 1:N search result of CASIA V3 Interval database for an eye from CASIA V1 showing images and segmentation boundaries.	208
Figure 4.26 – Sectors used by Lin <i>et al.</i> for determining the Iris location marked in black on image 001_1_1 from CASIA V1 dataset.	210
Figure 4.27 – Iris 019_1_1 from CASIA V1 (a) segmented by Shamsi-Lin method (b) average shrinking squares applied to image 019_1_1 to find the pupil centre.	212
Figure 4.28 – An example of an incorrectly segmented CASIA V1 image (a) and an example of a segmented image with one incorrect boundary (b).	213
Figure 4.29 – Image 0000_012 from the CASIA V2D1 image dataset with an elliptical border added to mask darker areas from interfering with the pupil detection.	215
Figure 4.30 – CASIA V1 image segmented by Lin <i>et al.</i> ’s method.	217
Figure 4.31 – Generic platform segmentation panel.	219

Figure 4.32 – Generic platform normalisation panel	220
Figure 4.33 – Example edge detection images.....	221
Figure 5.1 – Sample images with their respective canny edge images for low threshold of 0.08, high threshold of 0.15 and sigma 2.5. Each image has its edge image overlaid (a) CASIA V2D1 image 0002_008 and (b) CASIA V1 image 001_1_1.....	242
Figure 5.2 – Images from CASIA V1 segmented by Segment One	245
Figure 5.3 – (a) Image 001/02 from the IITD image dataset and (b) its Otsu threshold image.....	247
Figure 5.4 – MATLAB Canny edge detection of image 001/02 from the IITD image dataset.....	248
Figure 5.5 – Figure 5.4 logical AND'ed with Figure 5.3b.....	248
Figure 5.6 – The left image is fionar1 from the MMU V1 image dataset segmented with the first version of Segment Two, the right image is the monochrome threshold image. .	249
Figure 5.7 – The left image is the Canny edge image of fionar1, the right image is the edge image logical AND'ed with the threshold image in Figure 5.6.	249
Figure 5.8 – Images from MMU V1 image dataset segmented with Segment Two using Lin <i>et al.</i> 's iris detection and Masek's eyelid detection.	255
Figure 5.9 – Image 0002_002 from CASIA V2D1 gamma corrected with boundaries & sectors marked on it.	257
Figure 5.10 – Original iris sectors from Figure 5.9.....	257
Figure 5.11 – Intensity stretched Figure 5.10 images	257
Figure 5.12 – Gaussian smoothed Figure 5.11 images	257
Figure 5.13 – Sums of columns for both the left (a) and right (b) sectors respectively. Y-axis indicates sum, x-axis indicates x position within the original sector	258
Figure 5.14 – The gradients of the column sums for both the left (a) and right (b) sectors. Y-axis indicates gradient, x-axis indicates x position within the original sector.....	259
Figure 5.15 – CASIA V2D1 image 0002_002, segmented using Segment Two.....	259
Figure 5.16 – Example edge image showing the connected and unconnected edge locations	263
Figure 5.17 – Comparison of true match results from Segment Three against the true match results attained by Segment One, Segment Two, Masek, OSIRIS and Li's segmentation algorithms.....	267
Figure 5.18 – Generic platform results pane showing the reference segmentation for image 010204 from the MMU V2 image dataset.	272

Figure 5.19 – Automatic segmentation analysis for Segment Three with the CASIA V2D1 image dataset.....	273
Figure 5.20 – Automatic segmentation analysis for Lin <i>et al.</i> 's segmentation method with the CASIA V2D1 image dataset.	273
Figure 5.21 – Daugman's rubber sheet model [77], translating an elliptic iris from Cartesian to polar coordinates.....	276
Figure 5.22 – Daugman's rubber sheet model [77, 31] providing pupil displacement correction showing exaggerated pupil displacement for illustration.....	277
Figure 5.23 – The results from Flow One compared with those obtained using Segment Two and Three, OSIRIS, Masek and Li's segmentation algorithms.....	292
Figure 5.24 - A vector rotated using Masek's method.....	299
Figure 5.25 - An extended vector using Match One, with sliding window indicated.....	299
Figure 5.26 – Average time to find the Hamming distance between two iris codes.....	304
Figure 5.27 – Compare Result Databases GUI.....	306
Figure 6.1 – Image 010204 from the MMU V2 image dataset, segmented by Segment Three (a) with a fixed gamma correction of 2.2 applied before edge detection, (b) without any gamma correction.....	322
Figure 7.1 – Histogram for bitmap image 001_1_1 from CASIA V1 dataset showing the large peak at intensity 41.....	325
Figure 7.2 – Histogram for image 001_1_1 from CASIA V1 against S1143R01 from CASIA V3i.....	326
Figure 7.3 – CASIA V2D1 images segmented by Lin <i>et al.</i> 's method within the generic platform.....	328
Figure 7.4 – Four accumulators from the three dimensional Hough matrix created by segmenting the CASIA V1 image 001_1_1.....	332
Figure 7.5 – Two accumulators (a) showing widely disparate maxima, (b) showing a focussed maxima at the centre.....	333
Figure 7.6 – Perpendicular Bisectors.....	335
Figure 7.7 – Extracting the Iris GUI zip file using the mouse in Windows 7.....	350
Figure 7.8 – The zip file extraction dialog window in Windows 7.....	351
Figure 7.9 – The 'Set Path...' menu option in MATLAB.....	352
Figure 7.10 – The 'Add Folder' button in the 'Set Path' dialog box.....	352
Figure 7.11 – The 'Move to Top' and 'Save' buttons in the 'Set Path' dialog box.....	353
Figure 7.12 – An example image database.....	354
Figure 7.13 – The dialog box for selecting the image database location.....	354

Figure 7.14 – The dialog box for selecting the result database location.....	355
Figure 7.15 – The generic platform GUI in Automatic Mode	356
Figure 7.16 – The generic platform GUI in Manual Mode showing the results of a segmentation operation	358
Figure 7.17 – The generic platform GUI in Manual Mode showing the results of a normalisation operation.....	359
Figure 7.18 – The generic platform GUI in Manual Mode showing the results of an encoding operation	360
Figure 7.19 – The generic platform GUI in Automatic Mode processing the selected image dataset with the selected iris recognition algorithms	361

List of Tables

Table 3.1 – Summary of the image datasets available to this research, the features of each of the datasets and their technical details.....	116
Table 4.1 – Iris recognition algorithms available for download.....	140
Table 4.2 – Test machines used to develop and evaluate the generic platform and iris recognition algorithms.....	141
Table 4.3 – CASIA V1 results, Masek’s claimed results versus generic platform (GP) results in both ‘fixed flow’ and ‘custom flow’ form.....	147
Table 4.4 – Actual image datasets used by Masek.....	149
Table 4.5 – Masek’s claimed CASIA V1 results with generic platform (GP) results for the reduced 633 and 624 image CASIA V1 datasets and the full CASIA V1 results.	150
Table 4.6 – Results from Masek’s algorithm with the remaining test datasets.....	155
Table 4.7 – Results of testing Masek’s flow with the segmentation mask thresholding removed.....	161
Table 4.8 – Results of Masek’s flow with a fixed segmentation mask threshold of 75.....	162
Table 4.9 – Results of Masek’s flow with a fixed segmentation mask threshold of 50.....	163
Table 4.10 – Results of Masek’s flow with a fixed segmentation mask threshold of 25...	164
Table 4.11 – Processing times from using Masek’s flow without thresholding. (s) seconds, (h:mm) hours and minutes.	166
Table 4.12 – Results of Antonino’s flow without a fixed segmentation mask threshold. .	174
Table 4.13 – Processing times from using Antonino’s flow without thresholding. (s) seconds, (h:mm) hours and minutes.....	176
Table 4.14 – The results obtained over ten runs when using Li’s flow with the CASIA V1 image dataset.....	180
Table 4.15 – The results obtained over ten runs when using Li’s flow with the WVU Free dataset.....	181
Table 4.16 – Average results of ten runs of Li’s flow with the test datasets within the generic platform from Table 4.14 and Table 4.15.....	182
Table 4.17 – Processing times from using Li’s flow for the two datasets that were completed. (s) seconds, (h:mm) hours and minutes.....	183
Table 4.18 – Results of Li’s segmentation and normalisation algorithms with Masek’s encoding and matching routine.	184
Table 4.19 – Results of Li’s segmentation algorithm with Normalise One and Masek’s encoding and matching routines.....	185

Table 4.20 – Results from the JIRRM segmentation algorithm with Masek’s normalise, encode and matching routines	188
Table 4.21 – Processing times for the JIRRM algorithm with the remaining test datasets. (s) seconds, (h:mm) hours and minutes.....	190
Table 4.22 – Recognition results from the JIRRM segmentation algorithm using Masek’s upper and lower pupil and iris circle sizes with Masek’s normalise, encode and matching routines.....	191
Table 4.23 – Processing times from the JIRRM algorithm looking for a wider range of circle sizes for both iris and pupil with Masek’s normalise, encode and matching routines. (s) seconds, (h:mm) hours and minutes.....	193
Table 4.24 – Processing times for the C++ OSIRIS executable, total time in hours, minutes and seconds, average time in seconds	195
Table 4.25 – Recognition results from the OSIRIS segmentation algorithm with Masek’s normalise, encode and matching routines.	197
Table 4.26 – Results of Masek’s algorithm using various edge detection methods for the iris boundary detection on CASIA V1 dataset.....	222
Table 4.27 – Results of Masek’s algorithm using various edge detection methods for the pupil boundary detection on CASIA V1 dataset.....	223
Table 4.28 – Results of Masek’s algorithm using the JIRRM Hough for the iris and pupil boundaries in turn.....	225
Table 4.29 – Total times from running Masek’s algorithm as a ‘fixed flow’ and as a custom flow in MATLAB 2009a x64 on Windows 7 (PC1) against the CASIA V1 and V3i datasets	228
Table 4.30 – Test machines used to develop and evaluate the generic platform and iris recognition algorithms	230
Table 4.31 – Average processing times over five runs of using Segmentation Three and Normalise One with Masek’s encoding and matching code on CASIA V1 dataset.....	231
Table 4.32 – Average processing times over five runs of using Segmentation Three and Normalise One with Masek’s encoding and matching code on PC1, PC2 and PC3 using the CASIA V2D1 dataset.....	231
Table 4.33 – Average processing times over five runs of using Segmentation Three and Normalise One with Masek’s encoding and matching code on PC1, PC2 and PC3 using the CASIA V3i dataset.....	232
Table 4.34 – Time for code creation using Masek’s algorithm on the CASIA V1 dataset, without and with pre-calculation on PC1	234

Table 5.1 – Recognition results from Segment One using Normalise One, Segmentation Mask One, Masek’s encode and Match One.....	243
Table 5.2 – Code creation times of Segment One using Normalise One, Segmentation Mask One and Masek’s encode.....	244
Table 5.3 – Pixel count and averages for final edge image, without gamma correction, with fixed gamma correction and using gamma correction with adaptive intensity thresholding	250
Table 5.4 – Recognition results from Segment Two using Normalise One, Segmentation Mask One, Masek’s encode, Lin <i>et al.</i> ’s iris boundary detection and Match One	253
Table 5.5 – Code creation times of Segment Two using Normalise One, Segmentation Mask One and Masek’s encode.....	254
Table 5.6 – Recognition results from Segment Two using the described iris boundary detection, Normalise One, Segmentation Mask One, Masek’s encode and Match One....	260
Table 5.7 – Recognition results achieved with Segment Three using Normalise One, Segmentation Mask One and Match One with Masek’s encoding algorithm.....	266
Table 5.8 – Code creation times attained using Segment Three with Normalise One and Masek’s encoding algorithm	268
Table 5.9 – Recognition results from Segment Three, Normalise One with the intelligent infilling algorithm, Match One and Masek’s encoding routine	283
Table 5.10 – Recognition results from Segment Three, Normalise One with the OSIRIS infilling algorithm, Match One and Masek’s encoding routine	284
Table 5.11 – True match results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.....	286
Table 5.12 – Separability results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.....	286
Table 5.13 – Decidability results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.....	287
Table 5.14 – EER results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.	287

Table 5.15 – Recognition results from Normalise One using oversampling with the developed iris boundary detection, Segmentation Mask One, Masek’s encode and Match One	290
Table 5.16 – Recognition results from Normalise Two using the combined normalisation improvements with Segment Three, Segmentation Mask One, Masek’s encode and Match One	293
Table 5.17 – Time to analyse each dataset using Masek’s Hamming match (hours:minutes)	297
Table 5.18 Comparison of the difference calculations versus the HD. A and B are the input data A - B is the magnitude of the difference calculation, $A \oplus B$ is the logical XOR of A and B in binary, and the last column shows the HD bit count result from the $A \oplus B$ column.....	302
Table 5.19 – Analysis times using Masek’s Hamming match versus Match One.....	303
Table 6.1 – Summary of average recognition results and average code creation time per image for all the algorithms tested.....	317
Masek’s results in Table 6.2 are highlighted as they were used as the benchmark during testing. Li’s results are shaded because the averages shown only include results from the two image datasets that Li’s algorithm completed processing on, thus they do not represent a fair comparison to the other results. The results from Lin, Lu and Shamsi-Lin are shaded because these algorithms did not successfully segment images so the results do not present a useful comparison of capability.	317
Table 6.3 – Comparison of the total analysis time when using Match One as compared to using Masek or Antonino’s algorithms	318
Table 7.1 – Description of the generic platform GUI regions from Figure 7.15	357

List of Image Datasets

Table 1 – The iris image datasets available to this research

Dataset	Resolution (X × Y)	Number of Images	Number of Classes
CASIA V1	320×280	756	108
CASIA V2D1	640×480	1,200	60
CASIA V2D2	640×480	1,200	60
CASIA V3i	320×280	2,639	395
CASIA V3L	640×480	16,625	819
CASIA V3t	640×480	3,284	400
CASIA V4d	2352×1728	2,567	284
CASIA V4t	640×480	19,980	2,000
CASIA V4s	640×480	9,990	1,000
IITD	320×240	2,240	224
MMU V1	320×240	450	90
MMU V2	320×238	995	199
UBIRIS V1	800×600	1,249	241
UBIRIS V2	400×300	11,101	522
UPOL	786×576	384	128
ICE 2005	640×480	3953	244
WVU	640×480	865	200
WVU Free	640×480	80	20

Full descriptions of the image datasets listed in Table 1 are provided in Section 3.7.

List of Symbols

$*$	Convolution operation.
\cap / AND	Logical, bitwise AND operation
a	Wavelet size
α	Square of the vector length between two circle centres
β	Angle of the vector between two circle centres
b	Wavelet size
$CodeA$	Iris code
$CodeB$	Iris code
$C(u)$	Discrete Cosine Transform (DCT) result coefficients
d'	Decidability
$f(j)$	Input signal
$G(x, y)$	Gamma corrected image
$G_\sigma(r)$	Smoothing function over the radius r . Typically, this is a Gaussian smooth
HD	Hamming Distance
$I(x, y)$	Input greyscale image in Cartesian form
$I(\rho, \phi)$	Translation-invariant, dimensionless polar iris image

Im	Imaginary portion of a complex number
j, u	Index variable
$\ \cdot \ $	Mathematical norm
μ_S	Intra-class mean
μ_D	Inter-class mean
$MaskA$	Iris mask
$MaskB$	Iris Mask
N	Data length or count
ω	Angular frequency ($\omega = 2\pi f$)
\cup / OR	Logical, bitwise OR operation
Ox	Delta between two circle centres in the x direction
Oy	Delta between two circle centres in the y direction
ϕ	Phase offset
π	Pi, the constant mathematical ratio of a circles circumference to its diameter
r	Radius of a circle
Re	Real portion of a complex number
ρ	Radial distance from a centre point
σ	Standard deviation

σ_S	Intra-class standard deviation
σ_D	Inter-class standard deviation
θ	Angular offset
x_0, y_0	Cartesian coordinates for the centre of a circle
\oplus / XOR	Logical, bitwise exclusive-OR operation

Glossary

Authentic Accept (AA)	This is the status of an iris recognition system that has correctly matched a presented iris
Authentic Reject (AR)	This is the status of an iris recognition system that has incorrectly rejected a presented iris, this would count as a false reject in the FAR
CASIA	Chinese Academy of Sciences Institute of Automation. Released the first iris image dataset to the international community, CASIA-Iris V1, and have since updated this with CASIA-Iris V2, V3 and V4.
Class	The class of an image is the technical term for the source of the eye. If two images are from the same class, then they are both images of the same eye.
Code Creation Time	The time it takes an iris recognition algorithm to create an iris code from an input image. This will include the pre-processing, segmentation, normalisation and encoding processing steps.
Custom Flow	This is a partial or complete iris recognition process broken into four distinct executables or stages designed for the generic platform. Custom flows allow the selection of different segmentation, normalisation, encoding and/or matching methods for evaluation.
Decidability	This is a number that represents how well grouped the inter-class and intra-class results from a recognition algorithm are. This gives very similar information to separability but reduces the impact of outliers.
DNF	Did Not Finish. The test was not completed

Equal Error Rate (EER)	This is the percentage of FAR and FRR when the FAR is equal to the FRR. The EER can be a fast way of comparing the accuracy of different iris recognition algorithms as, typically, a lower EER means higher recognition accuracy.
False Accept Rate (FAR)	This is a measure of the percentage of irises that are accepted for being a match which are from different physical eyes. This is the opposite of separability and shows the number of false matches that have a Hamming distance lower than the highest true match Hamming distance
False Match	Two eye images that are confirmed as the best match but are not from the same physical eye. This is the opposite of true match, where the system is able to verify beyond reasonable doubt that the images are not of the class.
False Match Rate	The percentage of best matches for a given flow and dataset that are false matches, ideally this will be 0%
False Reject Rate (FRR)	This is a measure of the percentage of irises that are detected as being from different eyes when they are in fact images from the same physical eye
Fixed flow	This self-contained iris recognition process takes an image as its input and returns the segmentation, normalisation and encoding results. A 'fixed flow' will use a second executable that can perform matching on the results returned by the fixed flow.
Functions	These support-functions within a module are selectable within the GUI. Examples would be segmentation, normalisation, encoding, matching, edge, circle and line detection as well as segmentation mask generation.

GUI	Graphical User Interface. The screen objects with which a user interacts in order to operate a piece of software
Histogram	A graph showing the number of each colour or intensity value within an image. These graphs are used to establish colour composition and brightness levels in images among other things.
<HOSTNAME>	The configured hostname of the machine running MATLAB. This is usually configured by the system administrator during the OS installation.
ICE	The Iris Challenge Evaluation was an iris recognition competition organised by the American National Institute of Standards and Technology.
IIT	Indian Institute of Technology in Delhi. Released a set of clean forward facing iris images for the international research community to use.
Image Dataset	A collection of images stored in a directory on the computer's hard drive, for the purposes of recognition evaluation.
Inter-class	A statistical term indicating comparisons with items for different classes. The inter-class comparisons for iris recognition would be the comparisons of one eye image against all the images not from that eye.
Intra-class	A statistical term indicating comparisons within the same class. The intra-class comparisons for iris recognition would be a one-to-one comparison of all the images of the same eye.

Imposter Accept (IA)	This is the status of an iris recognition system that has incorrectly matched a presented iris, this would count as a false accept in the FAR
Imposter Reject (IR)	This is the status of an iris recognition system that has correctly rejected a presented iris
MMU	Multimedia University in Cyberjaya, Malaysia. Released two iris datasets, MMU V1 & V2, to the international research community.
Module	A module contains one or more flow(s) and/or custom flow(s), typically but not always by the same author, that add functions to the generic platform.
NA	Not Applicable. Given the rest of the results, this result is not relevant. E.g. if there are no true matches then any equal error rate or equal error Hamming threshold will be invalid as these numbers require at least one true match.
NaN	Not a Number. Normally MATLAB returns this if a division by zero has occurred.
NIR	Near infrared. This is a description of the light wavelength that the majority of freely available iris image datasets are captured at. Capturing in the near infrared spectrum allows melanin pigmentation to be bypassed, clearly showing the iris patterns.
NIST	The American National Institute of Standards and Technology organised the Iris Challenge Evaluation (ICE) which was an iris recognition competition aimed at improving understanding and improving standards within the iris recognition field.

OS	Operating System. The core system software used on a computer
ROC	Receiver Operator Characteristic. This graph is a visual characterisation of the trade-off between the FAR and the FRR. For commercial biometric security systems the matching algorithm performs a decision based on a threshold that determines how close to a template the input data needs to be for it to be considered a match. If the threshold is reduced, there will be fewer false rejects but more false accepts. Thus, a higher threshold will reduce the false accepts but increase the false rejects.
Separability	The separability percentage gives a measure of how well separated the results from a recognition algorithm are. It is the percentage of true matches with a Hamming distance lower than the lowest false match Hamming distance. The higher this number the better, ideally it will be 100%
Stage	This refers to the iris recognition steps segmentation, normalisation, encoding and matching.

Test Machines

Table 2 shows the machines that were used for testing during this research.

Table 2 – Test machines used for this research

Test Machines				
Name	PC1	PC2	PC3	PC4
OS	64-bit Win7	64-bit Win7	64-bit LinuxMint 10 (Gnome Ed.)	32-bit WinXP
CPU	Core i7-2600K @ 3.40GHz	Core2Duo T9900 @ 3.06GHz		Pentium 4 @ 2.60GHz
GPU	nVidia GeForce GTX 550 Ti	nVidia Quadro NVS 160M		ATI Radeon 9250
RAM	8GB	4GB		1GB
MATLAB Version	r2009a			r2006a & r2009a

True Match

Two eye images that are confirmed as being the best match and from the same physical eye. This information is rarely available to real security systems which only have the Hamming distance to go on.

To determine if the best matching eye from a set is a true match the system must be able to determine beyond reasonable doubt that the two images are of the class.

True Match Rate

The percentage of best matches for a given flow and dataset that are true matches, ideally this will be 100%

UBIRIS

Unconstrained Biometrics: Iris. This is the name of two iris image datasets, UBIRIS V1 and V2, released by Proença et al, whose goal was the furtherance of research into more challenging conditions for carrying out iris recognition.

UPOL

University of Palackého and Olomouc. Released an iris dataset to the international research community that was acquired with a high resolution optometric sensor.

WVU

West Virginia University released an Iris dataset with more challenging iris images that are a mixture of forward facing, off angle and off axis.

1 Introduction

The aim of this project was to develop a generic software platform that allows better understanding of the process of iris recognition, eases experimentation with existing iris recognition algorithms and eases development and evaluation of new algorithms in a consistent manner. The project was not aiming to produce a commercially viable system for personal identification.

An industrial partner or group normally sponsors the Engineering Doctorate (EngD), with the aim of producing a product or portfolio of products that resolve an industrial problem. This industrial input is an important part of the EngD, providing commercial motivation and a customer led focus to the project. Unfortunately, circumstances dictated that there was no industrial sponsor for this EngD work.

To compensate for this lack of industrial input, the work was carried out with the aim of publishing the generic platform as an open-source iris recognition development tool. In order to maintain the industrial theme, the project was strongly focussed on the development and testing of a tool that would be useful within a commercial environment for the development of iris recognition algorithms. The generic platform was developed hoping that it would become the standard tool for developing and evaluating iris recognition algorithms.

The aim is to release the generic platform upon completion of this work. Two websites will be used, the MathWorks MATLAB Central web site, <http://www.mathworks.co.uk/matlabcentral/> and Source Forge, <http://sourceforge.net/>. The code will be released under an open-source licence with a user guide. The platform is then to be advertised via email to researchers and companies in the field of iris recognition.

The next section discusses the background and motivation for this thesis.

1.1 Motivation

The search for a reliable, consistent and easy to test method of identifying an individual has long been a necessary part of human life with this applying to identifying both citizens and criminals. One method, which has gained much traction in recent years, is iris recognition. The human iris has been shown to be remarkably individual [1, 2] and is now considered to be one of the most accurate biometric methods that are available for the unique identification of individuals.

There are many advantages to using identification systems based around human biometrics, such as fingerprints, palm-prints, hand geometries, irises, and the retina. Human biometric identification is about the natural things that make up an individual, and is not about knowledge or possessions they may have. Using biometrics can make it much more difficult for imposters to impersonate an individual, particularly when compared with simply requiring knowledge of a password or date of birth. Research has shown that even identical twins have different iris patterns [3].

The analysis of the human iris by a machine is typically broken down into the following distinct steps:

1. Image Capture
2. Image Pre-Processing
3. Image Segmentation
4. Image Normalisation
5. Iris Encoding
6. Iris code matching

In any system that is used for iris identification, the image capture step is normally carried out using a camera. Sometimes this camera is a still-capture device, and sometimes it is a video camera. For researchers and developers this step is normally handled by using pre-captured images. Several datasets are available from these sources:

- Chinese Academy of Sciences Institute of Automation (CASIA) [4][5][6][7]
- Multimedia University (MMU) [8]
- Unconstrained Biometrics: Iris (UBIRIS) [9][10]

- West Virginia University (WVU) [11, 12]
- Indian Institute of Technology (IIT) Delhi [13, 14]
- University of Palackého and Olomouc (UPOL) [15, 16]
- Iris Challenge Evaluation (ICE) 2005 [17]

The second stage of iris recognition, image pre-processing, is any alteration to the original image that helps to improve the recognition process. This can be as simple as adjusting the contrast within the image, to more complex tasks such as detecting and removing specular reflections within the image. Image pre-processing can also occur at different stages within the recognition process in order to maximise the effectiveness of that stage. For example, a strong blur combined with reflection infilling and intensity stretching might be the pre-processing for the segmentation of the iris boundaries, but for the normalisation stage only the infilling would be used.

The third stage, segmentation, finds meaningful segments or regions in the given images. It uses features of interest within the image to achieve this. For iris recognition, the features of interest are the pupil-iris and iris-sclera boundaries. Typically, researchers are also interested in discovering any eyelid boundaries to see if the eyelids are occluding any part of the iris.

The normalisation step follows segmentation and extracts the meaningful regions found during the segmentation process from the overall image, preventing any unwanted data from being processed in later stages. For iris recognition, the region of interest is the iris, which is extracted from the rest of the image to allow the next stage of encoding to focus on the iris texture. The normalisation process for iris recognition can also be used to remove sources of noise such as eyelids, eyelashes and specular reflections from the iris image. [Note that “noise”, in this work, means unwanted parts of the image, rather than the more normal meaning of random fluctuations, such as electrical or thermal noise.]

The fifth stage, encoding, takes the normalised iris image and converts it to a form that is suitable for the matching stage. For iris recognition, this will mean extracting the iris texture information from the iris image and codifying that in the form expected by the matching stage.

Finally, the matching stage compares the input code with all the stored codes in order to find the code with the highest degree of similarity to the input code.

These iris recognition stages are present within all the commercially available biometric systems that use iris recognition for identification.

The number of commercial systems in use has risen steadily in recent years as the field of iris recognition has increased its credibility. In the last decade the US, UK and United Arab Emirates (UAE) have all used, or are currently using, iris recognition to identify potential fraudsters and known undesirables, which clearly shows its increasing role in security applications.

The biggest project to use iris recognition to date is the Indian government's Aadhaar project [18] which aims to collect the iris patterns and fingerprints of every individual in India. The project aims to help the poor of India get better access to benefits, especially in very rural areas where there is often no access to banking infrastructure.

There are also humanitarian efforts using iris recognition in the form of services like The Child Project™ [19], which aims to help find and positively identify missing children and adults.

All of the projects mentioned so far work at close distances of less than 30cm with consistent, controlled lighting conditions. The eventual goal for iris recognition is to be able to capture and process irises at any distance, from three centimetres up to 20 metres, under any natural or synthetic lighting conditions. Research is still being carried out in this area most notably by Proença *et al.* [20, 21, 22, 23] and Matey *et al.* [24].

To help researchers who are working in this less constrained environment, Proença *et al.* have released the UBIRIS image datasets [9, 10, 25]. The images from other iris image datasets such as CASIA [4, 5, 6, 7] and IIT Delhi [13], are captured at less than 30cm, taken using near infra-red (NIR) light. In contrast, the UBIRIS datasets are both full colour. The UBIRIS V1 dataset was captured at 10cm but the V2 was captured at between eight and four metres, from subjects walking a set path looking in different directions.

At the time of writing, Daugman's algorithms are believed to underpin all commercial implementations of iris recognition [3, 26]. However, commercial systems use closed algorithms with all test data being kept internally. Even published papers seldom give full details of their algorithms, with only a few publishing their code [27, 28, 29] for independent evaluation.

Despite the popularity of iris-recognition, there is very little in the way of clear data on the testing and accuracies of these systems, and very little in the way of comparative data between the various techniques employed to carry out the analysis. There is also a lack of consistency in the presentation of iris recognition test results [2, 30, 31, 22], making direct comparison between methods difficult.

There are also very few publicly available tools to aid in the development and evaluation of iris recognition algorithms. This means that every developer and researcher must write every part of the algorithm from segmentation to matching, as well as all the code to handle one-to-one matching, analysis and storage of the results.

The generic platform was developed to address these shortcomings and to improve the process of developing new algorithms for iris recognition. Specifically this thesis set out with the following objectives:

- To produce a unique, flexible, easy to use, easily extendable software platform that enables iris recognition developers to experiment with a variety of algorithms on real data (Chapter 3)
- To use the platform to develop novel, efficient iris recognition algorithms and compare these to the current state-of-the-art (Chapter 5)
- To compare the performance of algorithms across image datasets (Chapters 4 and 5)
- To create a system capable of improving the reliability and consistency of reported iris recognition results (Chapter 4)

The next section outlines the contents of this thesis.

1.2 Thesis Outline

A review of iris recognition literature is presented in Chapter 2. This includes a brief history of iris recognition and the structure of the human eye. This is followed by a review of published literature in each of the areas of image capture, image pre-processing, segmentation, normalisation, encoding and matching. Finally, a very brief description is given of iris recognition algorithms that are available for testing.

Chapter 3 describes the various design decisions that were made during the work and the reasoning behind them. The chapter starts with a discussion of the programming languages that could be used for the development of the generic platform and then gives the reasons why it was finally determined to use MATLAB. There then follows a description of the wide variety of available formats for results, with a summary detailing which was selected for the generic platform, and why.

The next section of Chapter 3 discusses the ability of the platform to switch between different algorithms without recompilation or changing the main GUI code. This is followed by section a discussing the provision of built-in iris recognition algorithms to enable the user to develop algorithms for the individual stages of iris recognition without needing to develop any of the other stages.

The fifth section describes how the platform allows the user to switch between algorithms easily and the sixth section evaluates the process of adding an image dataset to the generic platform.

The seventh section in Chapter 3 contains a full list of all the datasets available to this research along with detailed descriptions of them. The names and features of each dataset are summarised and then the reasons behind the selection of the eight datasets used for testing in this work are explained. Finally, the finished GUI is introduced and its operation discussed.

Chapter 4 describes the evaluation of the generic platform. The chapter starts by testing Masek's open-source iris recognition code [31] without any modifications and then discusses how this code is included within the platform. Masek's work, which was only the

subject of an undergraduate thesis, is the most widely referenced open-source iris recognition algorithm in the literature, [32, 33, 34, 35, 36, 37] - quite an achievement.

Part of the reason for the success of Masek's work is that the source code for the iris recognition implementation outlined in his thesis was released open-source, so is free for anyone to use in any way that they would like. A C++ implementation of Masek's algorithm was even used to generate the baseline scores for the Iris Challenge Evaluation (ICE) competition organised by the American National Institute of Standards and Technology (NIST).

With Masek's algorithms integrated, the generic platform is then used to test his code using the CASIA V1 dataset, and the results are compared with those in Masek's thesis. The tests are then repeated using the remaining seven image datasets selected in Chapter 3 and the results explored.

The next section of Chapter 4 discusses the integration, testing and evaluation of each of the four remaining open-source iris recognition algorithms available at the time the research was carried out. Methods introduced by Li [12, 27], Antonino [38], Sutra *et al.* [39] and JIRRM [40] are converted into modules for execution and evaluation within the generic platform.

These four methods represented the state of the art within the open-source iris recognition field. Antonio's algorithm was a derivation of Masek's work, replacing just the encoding and matching algorithms. Li and Sutra's algorithms are 'fixed flow's containing all the recognition stages. JIRRM uses the Hough transform [41], which, in Java at least, executed very quickly. It was of interest to determine whether the algorithm would perform faster or slower than Masek's Hough transform code when run within the MATLAB environment.

The reasons for not converting the remaining three algorithms from Rosa [42, 43] and GruSoft [44] to run under the generic platform are then discussed.

Section 4.5 looks at how well the platform can aid the process of evaluating other researchers' work using, for testing purposes, three papers selected from the literature. In each case, only the segmentation method was added as this enabled direct comparison of the overall results as affected by the segmentation algorithm alone.

This is followed by the inclusion, testing and evaluation of seven edge detection algorithms and two Hough transform[41] algorithms as separately selectable support functions.

The final section of Chapter 4 examines the performance concerns with respect to the generic platform. The first of these was the integration of algorithms as a 'Custom Flow' as compared with a 'Fixed Flow'. The impact on performance of using MATLAB as the development environment is also discussed and options for improving this are set out along with a partial solution to the problem. Lastly, there is a comparison of the performance variations across different computers and operating systems.

Chapter 5 describes the algorithms developed during this research, Segment One, Two and Three, and the results obtained by using them. The first section describes and evaluates Segment One, which is a Hough transform based algorithm that uses Canny edge detection [45]. The novelty in this algorithm was the use of a single edge detection and a single Hough transform operation to find both iris and pupil boundaries.

The second section in Chapter 5 describes and evaluates Segment Two, which uses the Hough transform with Otsu thresholding [46] for the pupil detection. The Otsu thresholding helps minimise the number of points in the edge image, thus reducing the Hough transform processing time. The iris boundaries are determined using a sum-of-columns method. The novelty of this algorithm is the use of Otsu thresholding to help find the pupil and the combination of Gaussian smoothing with the horizontal Sobel convolution matrix with the sum-of-columns to determine the iris boundary.

Segment Three, in Section 5.3, uses a novel method of selecting edge points from a Canny edge image for use with the circle bisector method to find the pupil and eyelid boundaries. Averaging and weighting operations are then used to improve the accuracy of the circles discovered. The iris boundary is determined using a slightly improved version of the sum-of-columns method that was created for Segment Two, with additional Gaussian smooth operations.

An automatic segmentation evaluation algorithm is described that uses the results from all the segmentation algorithms tested during this research to evaluate the segmentation boundaries of algorithms under test. This appears to be a totally novel algorithm with no automatic methods of validating iris segmentation boundaries found in the literature.

There then follows a description of the two normalisation algorithms developed in this work, Normalise One and Two. The first, described in Section 5.5, replicates Masek's normalisation code but extends it to cope with elliptical iris boundaries and circular and elliptical eyelid boundaries. Others have produced similar algorithms, but this allowed Masek's normalisation to be used for testing with elliptical segmentation boundaries.

Normalise Two is discussed in the next section and shows how three modest, but novel, enhancements to the normalisation process give a significant improvement in performance in an area that is often neglected by researchers. The results obtained are compared with those from Normalise One.

Section 5.7 discusses a more flexible segmentation mask routine called Segmentation Mask One (SM1) that can be used to replace Masek's. This algorithm removes the thresholding present in Masek's segmentation mask and adds the ability to mask elliptical and circular eyelid boundaries.

Match One follows SM1 and shows how a simple algorithmic improvement can significantly increase the speed of matching. Given that it is simply an improvement in the implementation of an algorithm it has no effect on the matching performance. It was tested using Masek's segmentation, normalisation and encoding. The novelty in this algorithm is its use of integers to represent the iris codes. This is a coding device that is specific to the generic platform environment, but which improves performance considerably within that environment.

Match One and SM1 have been shown to have no measurable effect on the recognition results when using Masek's routines, but together they provide greater flexibility of boundary shape and significantly reduced analysis time.

The final section in this chapter details a small GUI that allows the comparison of results databases. This is very useful for detecting the presence of differences within sets of results, especially for 'Fixed Flow' against 'Custom Flow'.

The next short section summarises the outcomes for this thesis.

1.3 Thesis Outcomes

The key achievements outlined in this thesis can be summarised as:

- Creation of a generic platform that allows rapid development and testing of iris recognition algorithms. This platform may also be useable for other similar biometric process development. There is nothing even remotely resembling this tool in existence in the public domain.
- A comprehensive comparative evaluation of existing open-source iris recognition algorithms.
- Segment Two - A successful pupil segmentation algorithm that combines an adaptive intensity threshold with the Hough transform to return good recognition rates and very fast performance. The use of Otsu thresholding to create a mask for reducing the number of points on an edge image, thus increasing the performance of the Hough transform operation is unique within the field.
- Segment Three - A very fast, successful, segmentation algorithm that combines a novel data point selection technique with the circle bisector method and an effective sum-of-columns method to return very good recognition rates.
- A successful implementation of automatic segmentation evaluation. This is entirely unique, with no other attempts to automatically analyse segmentation performance recorded within the literature.
- Normalise Two - An improved normalisation algorithm that enhances the performance of the iris recognition process. The normalisation step is largely ignored within iris recognition, yet three unique changes, that individually give only modest improvements are combined to generate a significantly better overall improvement in recognition performance.
- Match One - A very fast matching algorithm specifically aimed at optimising the generic platform. This uses an optimised algorithm, pointers to shift masks and iris

codes and a unique conversion to integer values. All significantly improve matching and therefore analysis time.

So far, this work has resulted in this publication:

- *Conference paper in the 5th IAPR International Conference on Biometrics, ICB 2012, New Delhi, India, March 29 - April 1 – This is included in full in Appendix Appendix A*

The next chapter discusses the history of iris recognition and the state-of-the-art for each of the six steps of the iris recognition process listed and briefly discussed in Section 1.1. The chapter finishes with a look at available open-source iris recognition algorithms and determines if any are GUI tools to aid in the development of an iris recognition algorithm.

2 Literature Review

The iris starts developing in the third month of gestation and is complete by the eighth with only pigmentation changes, which recognition systems ignore, continuing into the first year after birth. Figure 2.1 shows an adult human eye, image 6_3 from the WVU Free image dataset, with each of the features of interest for the purpose of iris recognition labelled.

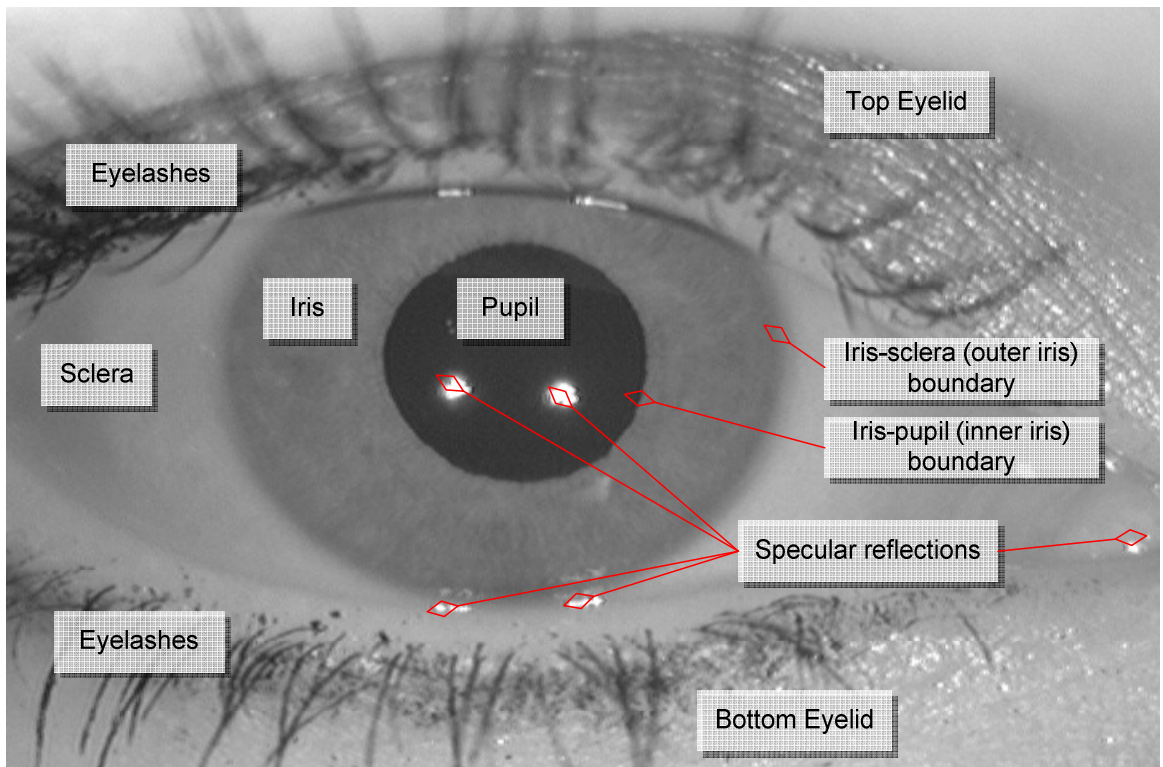


Figure 2.1 – Image 6_3 from the WVU Free image dataset, with each of the features of interest for the purposes of iris recognition

The WVU image, 6_3, in Figure 2.1 is labelled with the features of interest for iris recognition. The iris is the part that is sought, and is bounded by the iris-sclera and iris-pupil boundaries. The pupil, sclera, eyelids, eyelashes and specular reflections are all noise to be removed from the extracted iris image or at the very least, excluded from the matching process.

According to Wildes *et al.* [47] the first use of iris recognition for personal identification was in the late 1800's, when the iris colour patterning of inmates in a Parisian prison was inspected visually to determine their identity. However, it was the end of the Second World

War before ophthalmologists started writing more seriously about the possibility of using the iris patterns of the human eye as a method of identifying individuals. In 1953 Adler [48] wrote “...*the markings of the iris are so distinctive that it has been proposed to use photographs as a means of identification...*”. Of course, at that time, the technological ability to implement a real-time iris recognition system did not exist.

Nothing more happened with this idea until on 20th February 1985, when two ophthalmologists, Leonard Flom M.D. and Aran Safir M.D., applied for a patent for an Iris Recognition System. Over many years of clinical practice, they had observed highly detailed textures within irises that appeared to be unique for every eye in every individual. They could clearly see the variations in the iris patterns and this, combined with a review of the current literature, inspired Flom and Safir to apply for their patent, which the US patent office granted as US patent 4,641,349 on 3rd February 1987.

Flom and Safir had the patent but lacked the technical ability to implement any form of iris recognition, so they approached Dr John Daugman to help them develop the algorithms and technical methodologies necessary to turn their patent into a product. Daugman agreed to work with them and on 15th July 1991 Daugman filed for a patent with the US patent office for a “*Biometric personal identification system based on iris analysis*”. In the same year a report by Johnson[49] at Los Alamos National Laboratory, CA detailed initial work carried out to realise a personal identification system based on iris recognition.

Daugman proposed capturing images of the eye at very close range using a video camera and point light source. The system operator would then manually select the centre of the pupil. The close range reduces interference from ambient light and the manual selection speeds up and simplifies the segmentation process. Once the iris image was captured, a series of integro-differential operators, which show where the largest gradients are in the image, would be used to segment the iris.

The iris would be removed from the image and normalised to a rectangular form. This operation would remove artefacts, such as eyelashes, and other “noise”, and simplified the comparison process. This unwrapped iris image would then be encoded using two-dimensional Gabor filters, so converting the iris from a complicated multi-level bitmap image into a much simpler binary code. This encoded iris could then be compared to other

encoded irises by calculating the fraction of bits of the iris code that disagree, called the Hamming distance. More detail of these steps is given in Sections 2.1 to 2.6.

Daugman published two papers about his methodology, one in October 1992[1] and the other in November 1993[2]. On 1st March 1994 his US patent, 5,291,560, was granted allowing Flom, Safir and Daugman to setup a company and license the technology for commercial implementation.

The papers from Daugman instigated a new research area. In 1994 Daugman[50] further refined the ideas presented in his previous two papers[1, 2]. In the same year Wildes *et al.*[47] proposed a variation of Daugman's system using a wire reticule for alignment which he claimed was easier to use and potentially more accurate. Wildes also proposed the use of a diffuse light source to reduce specular reflections, a histogram approach to iris segmentation, which he claims is computationally simpler, and an alternative encoding and matching scheme that may make better use of the available information.

From that initial research by Daugman, a number of distinct steps for the analysis of the human iris by a machine were defined. Wildes followed these steps and they are still followed today by researchers and systems. The steps comprise:

1. Image Capture
2. Image Pre-Processing
3. Image Segmentation
4. Iris Normalisation
5. Iris Encoding
6. Iris code matching

There are further areas of relevance to iris recognition, live tissue verification, hardware acceleration, deployed applications and software platforms for the development and evaluation of new algorithms. Over the remainder of this chapter, the evolution of research in each of these areas is discussed.

2.1 Image Capture

Image capture involves creating a digital image of the eye being tested. The process of iris image capture must be non-invasive for the human subject under test, yet still be capable of capturing images of sufficient quality to evaluate the iris presented. At approximately 1cm in diameter, the iris is relatively small, sensitive to the light around it and humans are typically very protective of their eyes.

The images captured also need to be of sufficient sharpness and resolution for the recognition process and have a suitable level of contrast such that the iris pattern is clearly discernible without using annoying or damaging levels of illumination.

All of this effort is wasted, however, if the iris is not placed within the image in such a position that the algorithm can find it. In addition, the human subject should not be unduly inconvenienced by a need for over-exact positioning and multiple re-tries.

Because of these challenges, the history of iris image capture has become a story of distance and wavelength. That is distance from camera to subject and wavelength of the light used for capturing the images. Early research focussed on the use of greyscale, visible light video footage, captured at a range of less than 20cm to the subject.

Although Daugman's first paper on iris recognition[1] suggests the use of video as the image capture subsystem, his research at that point used a "*large photographic database of eye images*" provided by Safir and Flom. These were high quality, visible light still images, captured by ophthalmic cameras. Daugman's 1993 paper [2] expands this dataset to use pre-recorded VHS and S-VHS video footage manually filtered for quality and digitised via a computer video capture interface.

Daugman's system as uses live video and an LED-based point light source positioned below the end user [47]. Feedback is provided to the user using a liquid-crystal display (LCD) that is placed in-line with a beam-splitter that sends the image of the users eye to the camera, but allows the user to see the display, and thus what the camera is recording. A block diagram of Daugman's system is shown in Figure 2.2.

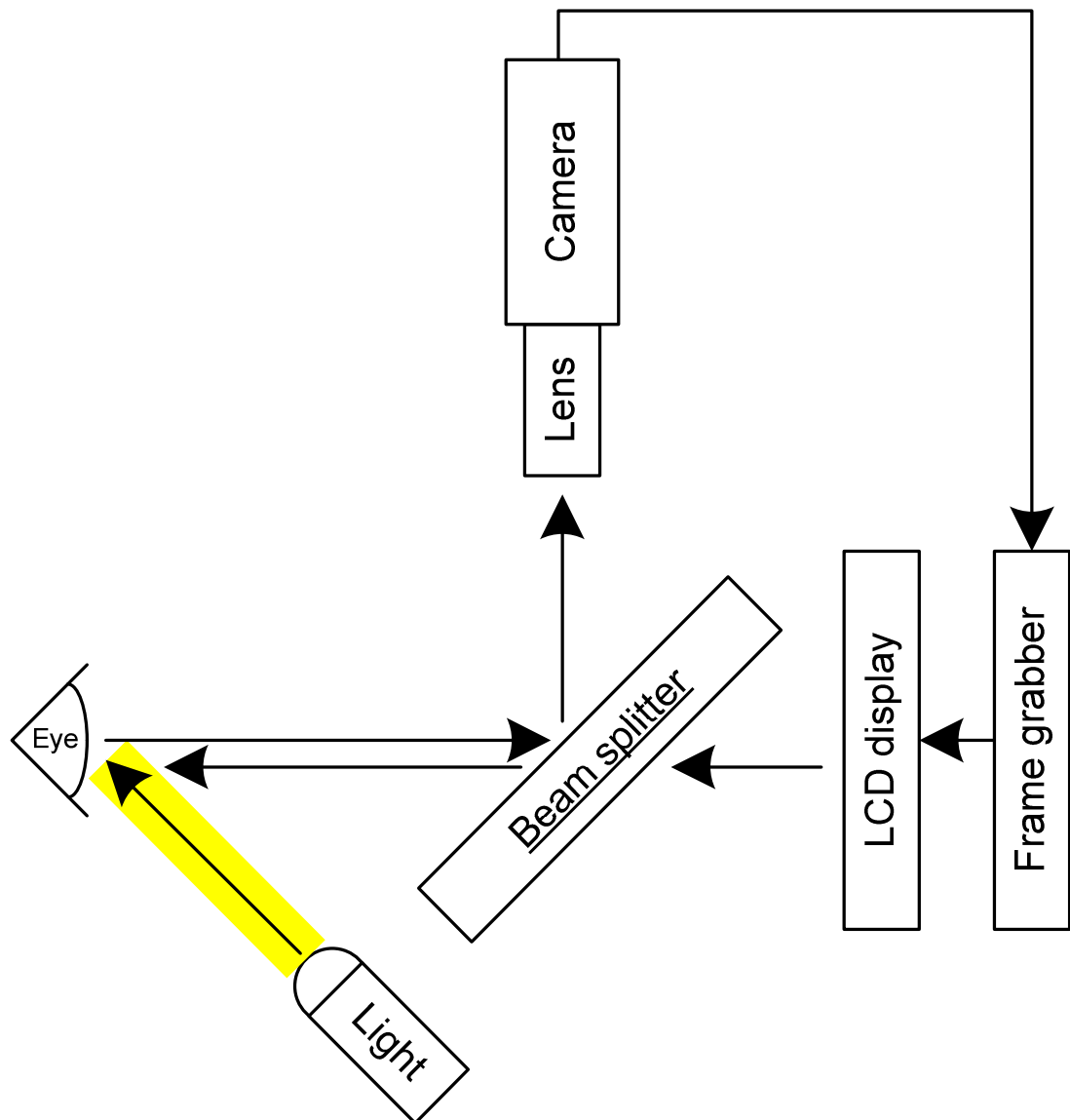


Figure 2.2 – A block diagram of Daugman's iris image-capture setup [51]

Wildes *et al.* use a diffused light source and high quality camera at 20cm from the subject. Their use of two light sources behind diffusers will significantly reduce the occurrence of specular reflections in the pupil and iris regions as compared to Daugman's equipment with the single point source of light in Figure 2.2. A block diagram of the system from Wildes *et al.* is shown in Figure 2.3.

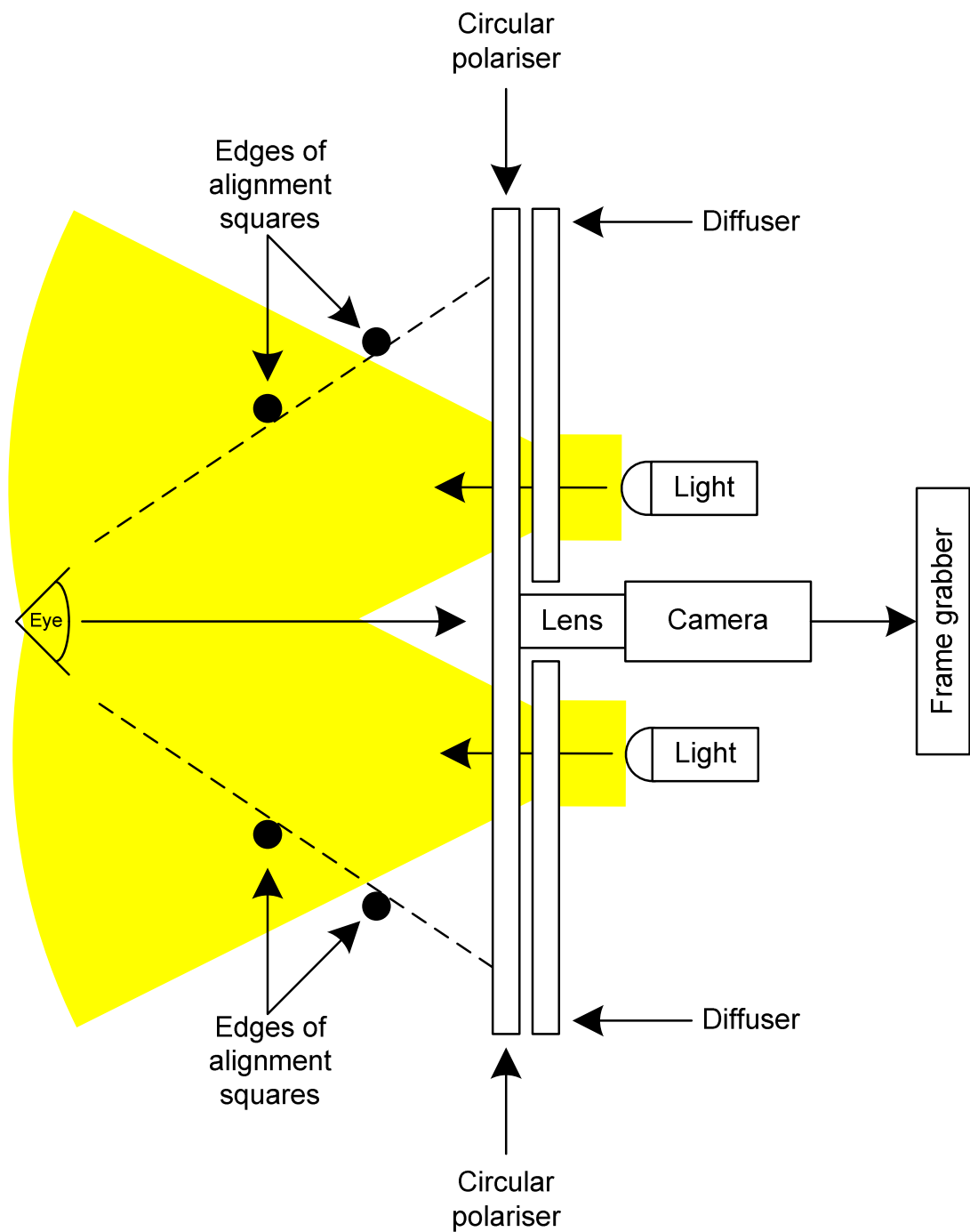


Figure 2.3 – A block diagram of the iris image-capture setup used by Wildes *et al.*[51]

Both these systems use the visible light spectrum, constrained targets and a range of 20cm or less to capture the iris images. By getting the subject to within 20cm of the camera (Figure 2.4), the eye can be isolated from the rest of the face and the surrounding environment making the task of isolating the iris much simpler.



Figure 2.4 – Example of a close-up iris image capture device being used [52]

While Daugman and Wildes *et al.* both use visible light, this is absorbed by the melanin in darker irises, making the determination of iris patterns for dark irises much more challenging. One solution to this problem is to use infrared light and cameras. At near infrared wavelengths the light passes straight through the melanin.

In their 1997 paper [51] Wildes *et al.* seemed unconvinced by the use of infrared simply stating, “*It has been suggested, however, that infrared illumination would also suffice*”. However, by 2003 when Libor Masek handed in his thesis [31] near infrared illumination and cameras were accepted as beneficial. This was most clearly demonstrated by the release of the first publically available iris image dataset, CASIA V1 [4], which used a near infrared camera and light source.

More recently, researchers have been focussing on capturing images in less constrained situations and over greater distances. The OKI IRISPASS [53] and LG iCAM [54] both aim to capture images at up to one metre. The Iris-on-the-Move project by Matey *et al.* [24], has been using near infrared imaging at one to two metres and Proença *et al* [23, 22, 25] have been looking at the visible wavelength at up to three metres.

Currently, there are many types of camera, lighting and distance in use for the capture process [52]. The use of video instead of still images can improve recognition rates for non-cooperative subjects [55, 56]. Once captured, the image(s) may be down-sampled, converted to greyscale and/or compressed. This post-processing all reduces image quality, though for compression, Carneiro *et al.*, Daugman, and Rakshit and Monro have shown

that even very strong lossy compression has little impact on recognition performance [57, 58, 59].

While this research does not look at the different imaging devices, it does use pictures from several different sources, produced using a variety of devices, and stored in both uncompressed bitmap and compressed JPEG image formats. The image datasets available to researchers that are more commonly used in the literature are:

- **Chinese Academy of Sciences, Institute of Automation (CASIA)** [4, 5, 6, 7]

The Chinese Academy of Sciences' Institute of Automation (CASIA) datasets are used throughout the world and have been referenced in many publications [3][30][31][60][61][62][63][35][64][65]. There are nine datasets containing 58,241 images for general research purposes. Each image is an 8-bit grey-scale image ranging in size from 320×280 pixels up to 640×480 pixels.

- **Multimedia University (MMU)** [8]

There are two MMU iris datasets; each collected using a different camera. The MMU datasets offer a good selection of ethnicities and image quality, so the algorithm under test is stressed, but, they contain only 1,445 images in total, not many when compared to CASIA's 58,241.

- **Unconstrained Biometrics: Iris (UBIRIS)** [9, 10]

The UBIRIS 1 dataset is similar to the MMU and main CASIA datasets in that it provides consistent close-up images of eyes in two different resolutions, as well as in colour and greyscale. The UBIRIS 2 dataset provides a much wider variety of image qualities and distances than the other iris image datasets. Taken together, these two datasets contain a significant 12,350 images.

- **West Virginia University (WVU)** [11, 12]

The West Virginia University iris dataset comprises 865 off-axis iris images, 268 of which were collected with a Sony Cyber Shot DSC F717 [66], while the remaining 597 were collected using a monochrome, black and white camera. There is a slight green hue to the Sony camera images as it was used in infrared mode, but still used all three RGB sensors.

The WVU dataset has quite stringent release pre-requisites and is only available on disc, not via download.

- **West Virginia University (WVU) Free** [27]

This sub-set of the WVU dataset is provided without restriction by Li [12, 27] alongside his iris recognition code download. It is a very small dataset with only 80 images, but contains images with a greater degree of variation of gaze direction than many others.

- **Indian Institute of Technology (IIT) Delhi** [13, 14]

The IIT Delhi iris dataset consists of iris images collected at IIT Delhi. This dataset was acquired using a JIRIS JPC1000 [67] digital CMOS camera, is from 224 users and all the images are in bitmap format. The users were between the ages of 14 and 55 years. The 1120 images are from 176 males and 48 females. All images are at a resolution of 320×240 pixels.

- **University of Palackého and Olomouc (UPOL)** [15, 16, 16]

This dataset contains 384 iris images (3×64 left and 3×64 right). The images are 24-bit, RGB, 576×768 pixels. The irises were scanned using a TOPCON TRC50IA [68] optical device connected to a SONY DXC-950P [69] 3CCD camera. The UPOL dataset is unusual in that a black border surrounds each iris. This can make detection difficult as there is a very sharp boundary outside the actual iris.

- **Iris Challenge Evaluation (ICE) 2005** [17]

This dataset contains 3953 greyscale iris images from 132 individuals captured in the near infrared (NIR) by an LG2200 camera. The images are of a single eye, with 1425 images of right-eyes from 124 individuals and 1528 images of left eyes from 120 individuals.

The image datasets are discussed in detail in Section 3.7. With the exception of the CASIA V4d dataset at 2352×1728 pixels, the images in these datasets are very low resolution when compared with the 2776×2096 resolutions found in modern phones and cameras, but effective iris recognition is still possible at these low resolutions.

After images are captured or loaded, in the case of image datasets, they are typically pre-processed in order to make the features of interest easier to detect. This pre-processing is discussed in the next section.

2.2 Pre-Processing

Having captured or loaded the eye image, the next stage is pre-processing. In machine vision applications, images are pre-processed in order to improve the ability of the machine to detect features and objects. Pre-processing can be as simple as intensity adjustments [70], such as intensity stretching, histogram equalisation and static gamma correction. It can also be more complex with thresholding [46][71], automatic gamma correction [72][73], noise removal [35] and edge detection [74][45][75][64].

Intensity stretching an image is a very simple but often effective processing step. The lowest and highest intensity in an image are not always the lowest or highest permissible values. In this situation, the contrast of the image can be improved by linearly adjusting the pixel intensities within the image such that they do stretch over the entire permissible range of values. This adjustment will make the shadows darker and the whites brighter, as well as giving the centre colours or shades greater separation.

Gamma correction is an operation similar to intensity stretching in that it adjusts the intensity of all the pixels within an image, but the adjustment is non-linear. Equation 2.1 is the calculation for gamma correction.

$$G(x, y) = I(x, y)^\gamma \quad 2.1$$

where $I(x, y)$ is the original image, γ is the correction value and $G(x, y)$ is the gamma corrected image

Equation 2.1's effect on the image depends on the value of γ . If γ is greater than one, then the darker intensities will be compressed so that their values are closer together, but the higher intensities will be stretched further apart. This will give an image where the brighter intensities are easier to discern but the darker ones become more difficult to separate.

If γ is less than one, then the brighter intensity pixels will have their values compressed together, while the darker intensity pixels will have their values stretched further apart. This will give an image that is brighter overall, where the brighter intensities may become difficult to discern, but the darker regions of the image will become much clearer and brighter.

Static gamma correction applies the same value of γ to every image, automatic gamma correction calculates the value of gamma to use on an image by image basis.

Since the earliest research into iris recognition by Daugman [1], Johnson [49] and Wildes *et al.* [47], researchers have realised the importance of pre-processing images to improve recognitions rates. The work carried out in this thesis does not investigate pre-processing in detail, however, initial investigations into intensity stretching, contrast adjustment and gamma correction showed that significant improvement in the recognition process could be achieved if these forms of image pre-processing were carried out. Pre-processing was used in the process of developing the second and third segmentation methods, discussed in Chapter 5.

Within the process of iris recognition, image pre-processing is different to the other stages. Even though it is placed within the process between image capture and segmentation, it may be carried out several times during and after the segmentation process.

2.3 Segmentation

Having pre-processed the image, the next step is to find the region of interest within the image, the iris segment. Image segmentation is the process of finding meaningful segments or regions in an image using features of interest. Segmentation for iris recognition involves identifying the pupil-iris and iris-sclera (limbic) boundaries as well as the top and bottom eyelid boundaries.

Figure 2.5 shows image S1004R05 from the CASIA V3i [6] image dataset that has been segmented using the Segment Three iris segmentation algorithm discussed in Section 5.3.

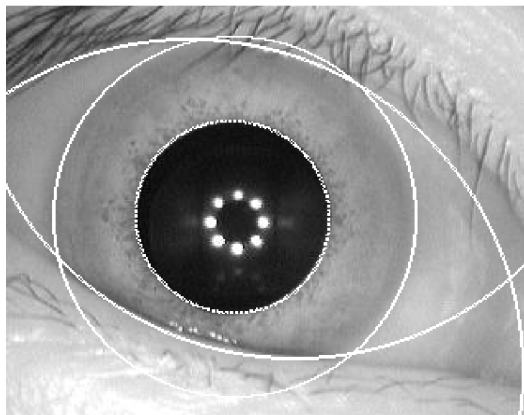


Figure 2.5 – CASIA V3i image S1004R05 with segmentation circles determined by Segment Three iris segmentation algorithm discussed in Section 5.3

Figure 2.5 has the boundaries detected by the segmentation algorithm overlaid in white. All four potential boundaries, iris-pupil, iris-sclera, top eyelid and bottom eyelid, have been detected in this image.

All of these boundaries can be treated as circles, and researchers from Daugman onwards have done this. This assumption stands up well for close images with cooperative subjects as the eye is very close to the axis between the camera and face. In these situations, the subject is looking directly at the camera and the iris-pupil and iris-sclera boundaries approximate to circles well enough for recognition to take place.

In order to find these circular boundaries researchers use edge detection routines that search for the largest changes in pixel intensity within an image. In 1993, Daugman [2] described his method of edge detection using integro-differential operators to search for the gradients in an iris image, as shown in equation 2.2.

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{r,x_0,y_0} \frac{I(x,y)}{2\pi r} ds \right| \quad 2.2$$

In equation 2.2, r is the radius of the integro-differential operation and x_0 and y_0 are the centre coordinates. $G_\sigma(r)$ is a smoothing function, for example a Gaussian smooth of scale σ which is convolved, for which the symbol is ‘*’, with the closed contour integral. The Gaussian smoothing convolution is a neighbourhood operation that takes each pixel in the original image and modifies its value to be a function of itself and the pixels surrounding it. σ determines the magnitude of the reduction in the standard deviation of pixel intensities across the image after the smoothing function is applied. $I(x,y)$ is the original image containing an eye.

The Gaussian smoothing function reduces the noise within the image, this reduces noise within the edge image so it is cleaner as fewer noise-related gradients will be detected. The surface integral within the integro-differential operation is implemented by summing the pixel values divided by $2\pi r$ along a radial line from the defined centre (x_0, y_0) , the result of which is the gradient at each pixel.

Figure 2.6 shows the integro-differential gradient image of the CASIA V3i image S1004R05 image in Figure 2.5, clearly highlighting the pupil boundary as well as the eyelashes, bottom eyelid and central specular reflections.

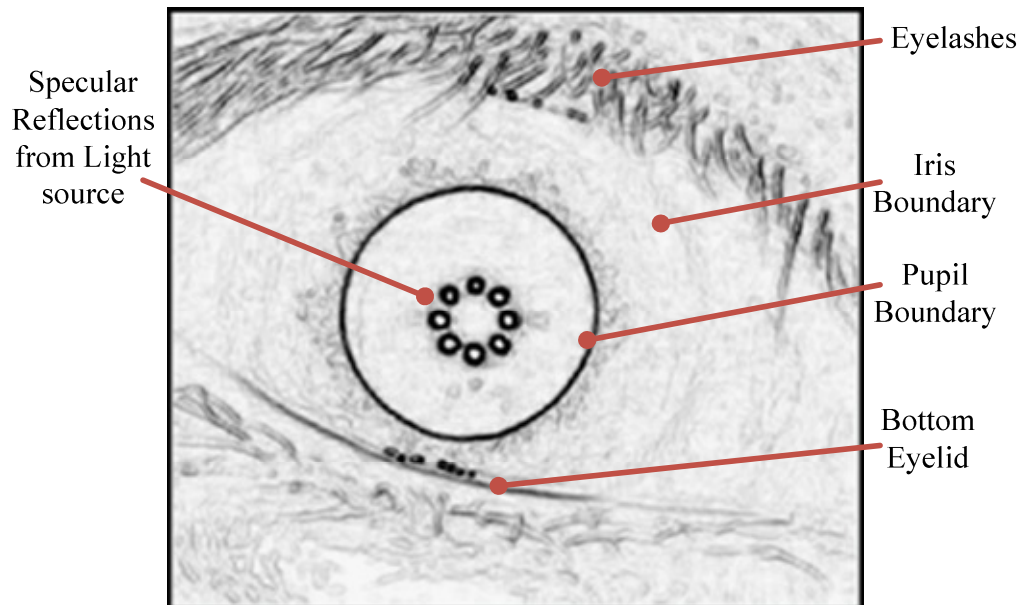


Figure 2.6 – Edge image resulting from the integro-differential operator being applied to CASIA V3i image S1004R05

One disadvantage of the integro-differential operator is that its effectiveness is strongly dependant on the defined centre (x_0, y_0) being within the pupil region the boundaries to be detected correctly. Daugman describes using multiple integro-differential operations to find the best centre location [76, 77, 78]

In 1994, Wildes *et al.* [47] suggested using a gradient-based edge detector with the Hough transform [41], and this was also used by Masek [31] in 2003. The Hough transform is described in more detail in Appendix Appendix C. The most popular gradient-based edge detection for iris recognition is the Canny edge detector. Originally published by John Canny [45], this edge detection, seen regularly in the literature, uses a multi-stage approach to detect edges in an image.

The first step in the Canny edge detector, as with Daugman's integro-differential operator, is Gaussian smoothing to reduce noise. The second step finds the gradients of the image intensities in the horizontal and vertical directions.

The edge image is then non-maxima suppressed, this means searching to see if the gradients are local maxima in the direction of the gradient. The output of this stage is a set of binary points showing the local maxima. Hysteresis thresholding is then applied to trace the edges, finally yielding the edge image. Figure 2.7 shows the edge image resulting from applying the Canny edge detector to image S1004R05 from the CASIA V3i dataset.

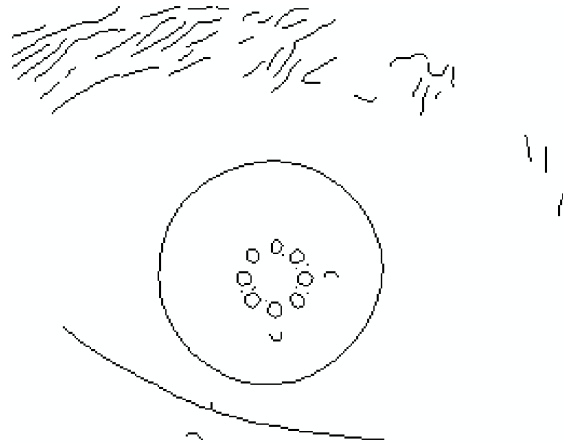


Figure 2.7 – Edge image resulting from the Canny edge detector being applied to CASIA V3i image S1004R05

Once the edges have been found, the edge image is searched for the desired boundaries. Several methods of shape fitting have been used for the curved boundaries of the eye. Daugman now uses active contours [76][78], others use circle fitting [30] like the Hough transform, ellipse fitting [35, 79, 80] or polynomial fitting [22]. The Hough transform is described in detail in Section 5.1. The work in this thesis focuses on using lines and circles as these have proven to be effective.

One challenge in the segmentation stage is the evaluation of its performance once the boundaries have been discovered. Evaluation of segmentation performance is normally carried out either by subjective manual evaluation [30][31][35] or by evaluation of the entire process results [38][3][81][60][82]. The only discovered attempt to automate segmentation performance in isolation was by the Nice.I [20] and Nice.II [21] contests, which employed a Java framework using manually classified data.

Segmentation of the iris boundaries is a very active area of research [62][3][52][30][37][31][35]. Many iris recognition research proposals

[83][3][57][1][30][29] have assumed a close up image of an eye (e.g. Figure 2.8) but latterly there has been a growing trend to look at whole face images [55][56] and non-cooperative subjects [84][85][25] for projects such as Iris-on-the-Move [52]. The work in this thesis concentrates on images taken at less than 20cm, as that is currently the most popular distance. Even when distances up to four metres are used, they finally focus on just one eye.

There are five proposals in the literature that all use images taken at 20cm or less and have similarities to Masek's work in particular. Lin *et al.* [30], Lu [60] and Gu *et al.* [86] all use the CASIA V1 dataset used by Masek and all use intensity thresholding as part of their segmentation algorithms. Lin *et al.* rely entirely on intensity thresholding for the pupil detection, Lu and Gu *et al.* use thresholding to identify points to use in conjunction with another step. Lu uses the circle bisector method for finding circles and Gu *et al.* use standard deviation. More details of Lin *et al.* and Lu's work are given in Section 4.5.

Aydi *et al.* [87] makes minor alterations to Masek's algorithm to achieve a 4% improvement in segmentation performance and 60% reduction in calculation time. These improvements are achieved by changing the type of interpolation used in the image resizing, altering the kernel used for the Gaussian blur and the edge detection method.

Shamsi *et al.* [62] use a novel method of average shrinking squares to detect the centre of the pupil in order to reduce the number of integro-differential operations needed to find the pupil boundary.

Once the boundaries have been detected, the next stage is to remove the iris from the image, thus eliminating the noise of the other parts of the eye and face from the process. This normalisation stage is discussed in the next section.

2.4 Normalisation

Once the iris boundaries have been detected in the segmentation stage, the normalisation stage physically separates the iris from the rest of the image and adjusts the spatial alignment of the iris into alignment with the candidate database. This isolates the signal from the noise and ensures that a like-with-like comparison occurs. Figure 2.8 shows image S1004R05 from the CASIA V3i [6] image dataset, segmented using the Segment Three iris segmentation algorithm discussed in Section 5.3.

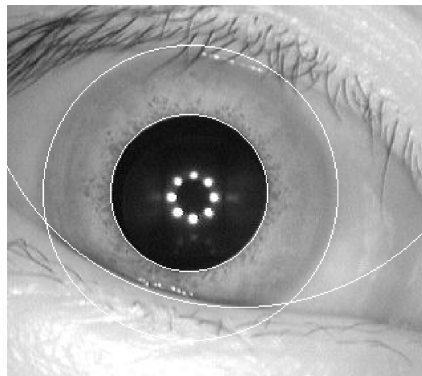


Figure 2.8 – CASIA V3i image S1004R05 with segmentation circles

Daugman [76] normalises the iris, based on its segmentation boundaries, by unwrapping it from its naturally polar state to a Cartesian form (Figure 2.9).

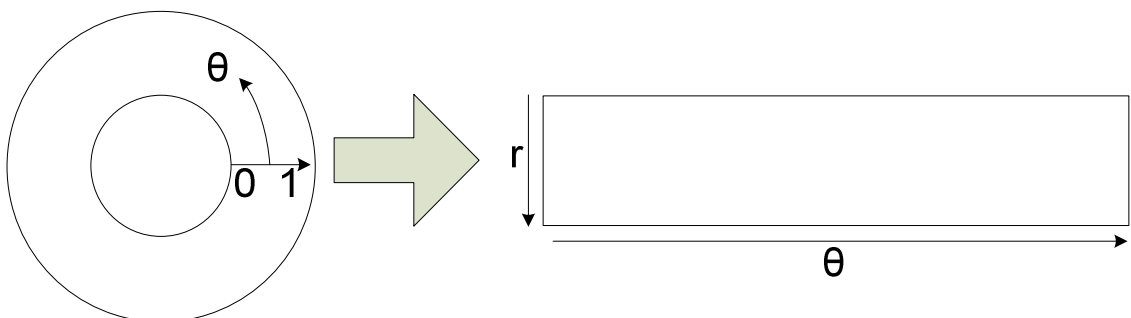


Figure 2.9 – Daugman's rubber sheet model[77, 31], translating from Cartesian to polar coordinates

The perfect translation shown in Figure 2.9 is not possible because the pupil is often not perfectly centred within the iris [78] and the dilation of the pupil will deform the iris between images. For this reason, Daugman treats the iris as a rubber sheet that stretches and deforms as its central pupil moves around and changes its radial size (Figure 2.10).

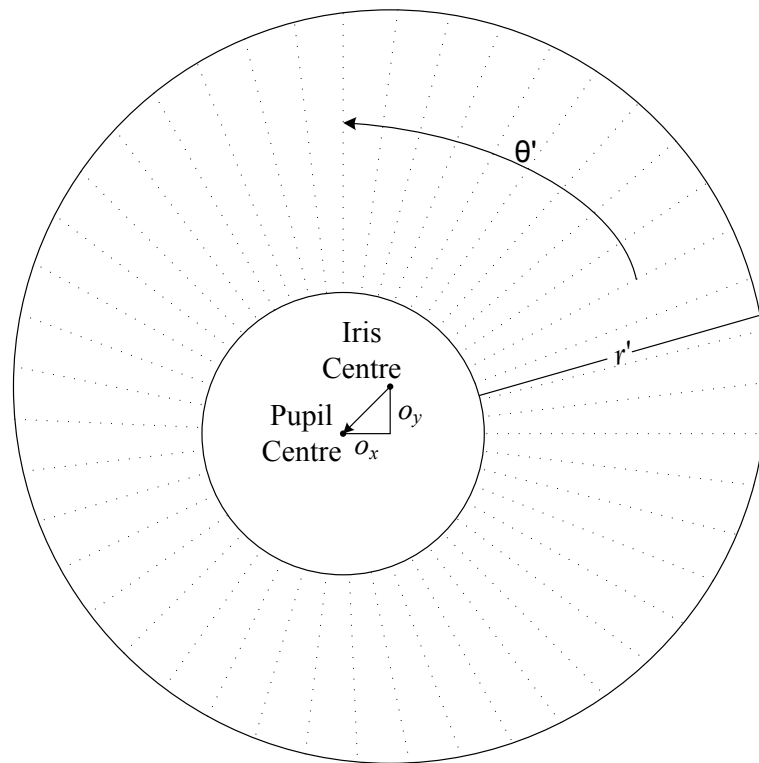


Figure 2.10 – Daugman’s rubber sheet model [77, 31] providing pupil displacement correction showing exaggerated pupil displacement for illustration

The pupil only changes dilation, not location from sample to sample, thus, as the radius of the pupil changes the radial sample points move in unison. This ensures that, when using the rubber sheet method, the same features should be captured each time the iris is acquired. By taking a consistent number of radial sample points at each value of θ but altering the inter-sample spacing to maintain even sample coverage across the iris, a consistent iris image can be constructed. The result of this ‘unwrapping’ process is shown in Figure 2.11, which shows the unwrapped iris pattern from Figure 2.8.



Figure 2.11 – Unwrapped iris pattern of Figure 2.8

Most modern implementations of Daugman’s ‘rubber sheet’ method also create a noise mask which is created during normalisation, updated during encoding and used during matching to exclude the noise pixels remaining within the normalised iris image [76, 88, 30, 31, 89, 28]. Figure 2.12 is the noise mask for Figure 2.11.



Figure 2.12 – Mask for Figure 2.11 unwrapped iris pattern

Daugman's rubber-sheet method is the standard method used to unwrap an iris image [60][3][63][55][52][37][31][35]. However, Birgale and Kokare [83] and Boles [90] proposed a method that leaves the iris in its original ring shape.

The normalisation step appears to be largely taken for granted in the literature, but this work investigates whether improvements in the recognition process can be achieved with only small changes in the normalisation process.

After normalisation has been completed, the final stage of the iris code creation, encoding, can be carried out, and this is the topic of the next section.

2.5 Encoding

The encoding stage takes the normalised iris image and extracts the discriminating features. Daugman [1, 2, 50, 91, 92, 81, 77, 76, 26, 78] proposed using 2D Gabor wavelets [93] to demodulate the normalised iris (Figure 2.11), extracting its phase information into a code that is fast for machines to process. The normalised iris is projected onto the complex-valued 2D Gabor wavelet [93, 94, 95, 96, 97] using equation 2.3.

$$h_{\{Re,Im\}} = \text{sign}_{\{Re,Im\}} \int_{\rho} \int_{\phi} I(\rho, \phi) e^{-i\omega(\theta_0 - \phi)} e^{-(r_0 - \rho)^2/a^2} e^{-(\theta_0 - \phi)^2/b^2} \rho d\rho d\phi \quad 2.3$$

where $h_{\{Re,Im\}}$ can be considered to be a complex-valued bit with real and imaginary parts that can be either 1 or 0 (sign), dependent upon the sign of the two-dimensional integral. $I(\rho, \phi)$ is the iris image in its translation-invariant, dimensionless, polar coordinate form, as shown in Figure 2.11.

The multi-scale two-dimensional wavelet size parameters are a and b which span a range of 0.15mm to 1.2mm on the iris. The wavelet frequency is represented by ω and is inversely proportional to β , spanning three octaves. The polar coordinates across the iris are represented by (r_0, θ_0) , while ϕ is the phase offset and θ is the orientation.

Note that while there is a superficial similarity between equation 2.3 and equation 2.2, there is no genuine relationship between them. Equation 2.3 demodulates the isolated iris in order to extract phase information using two dimensional quadrature Gabor wavelets in a dimensionless polar coordinate system, while equation 2.2 calculates gradient information radially in the original iris image using Cartesian coordinates.

In recent years the use of Gabor wavelets has become very popular in image analysis and computer vision, and this is no different for iris recognition [90, 78, 75, 61, 3]. Wavelets are brief oscillations, similar to the output of seismographs that begin at zero increase to a maximum and then return to zero.

The wavelet transform convolves this wavelet signal with the known signal, in this case the extracted iris, to extract the required information which, in the case of Daugman's iris encoding method, would be the phase information. Examples of three dimensional plots of

the wavelet patterns are shown in the top quadrants of Figure 2.13. Even though the integral produces real and imaginary numbers, the only information that is kept in the iris template is the quadrant that the numbers appear in on the complex plane, as shown in Figure 2.13.

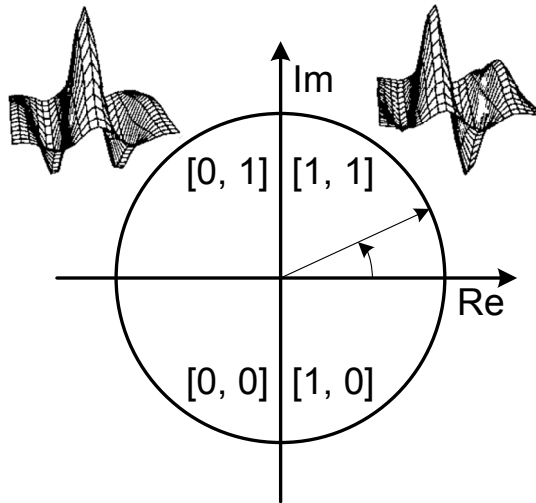


Figure 2.13 – Daugman’s phase quantisation code sequence [76]

Using this method, Daugman generates a 256-byte (2048-bit) digital code (Figure 2.14) which is unique for each iris pattern. Daugman selected this size for his code, because it fits in the space available on the magnetic strip of a credit card. The iris encoding process produces two bits for every pixel of the image, to accommodate the phase quadrature bit sequence shown in Figure 2.13.



Figure 2.14 – Daugman iris code of Figure 2.11

In 1994 Wildes *et al.* [47, 51] suggested that Daugman’s encoding scheme was not using all the information available and thus they proposed applying Laplacian of Gaussian (LoG) filters to the iris image. Wildes *et al.* use the filters shown in equation 2.4.

$$-\frac{1}{\pi\sigma^4} \left(1 - \frac{\rho^2}{2\sigma^2}\right) e^{-\rho^2/2\sigma^2} \tag{2.4}$$

where σ is the standard deviation of the Gaussian and ρ is the radial distance of a point from the filters centre. This is implemented as a Laplacian pyramid, shown in Figure 2.15.

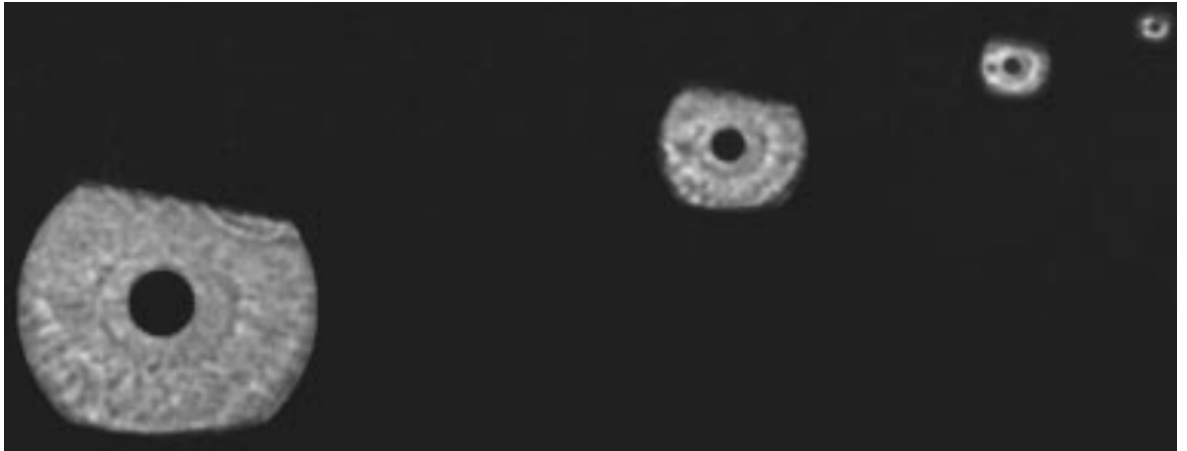


Figure 2.15 – Wildes multi-scale iris code [51] showing the results of successive sub-sampling of the lower frequency bands of the iris pattern

Figure 2.15 shows the results of successive sub-sampling of the iris pattern from the largest original isolated iris at the left of the image to the smallest which is used in the pattern matching process at the right of the image. Each time the iris pattern is sub-sampled the resulting iris pattern is one quarter the size of its predecessor.

Unlike Daugman, Wildes did not propose using phase quantisation, thus preserving more information for matching, at the cost of having to store and compare larger iris templates. Wildes' method generates an iris template that is directly related in size to the original image, making it less able to adapt to changing camera resolutions and variable capture ranges, due to the variation in source image size.

King and Phipps proposed Time Encoded Signal Processing and Recognition (TESPAR Coding) [98], as a method for digitally coding speech waveforms. The source signal zero crossings are discovered, and then the duration and number of troughs between zeros is noted and replaces the signal amplitude information between the zeros. This replacement of the signal with its duration and trough count (shape) between the zeros is termed 'infinite clipping'.

Emerich *et al* [88] presented a method of encoding that utilises the TESPAR DZ coding method instead of Daugman's phase quantisation method. TESPAR coding uses

approximations of the real and complex zeros derived from analysing waveforms that are band limited. Emerich *et al* use these zeros to encode the wavelet transformed iris. They also showed the effect on performance of using the various built-in MATLAB wavelet functions in the encoding process.

Hollingsworth *et al* [99] investigated the existence of fragile bits within an iris code. Bits can be defined as fragile if there is any probability of the bits varying in value between '1' and '0' between different images of the same iris. Hollingsworth suggested that during the encoding process, before the phase-quadrature encoding stage, masking any locations with complex numbers near the axes of the complex plane would reduce the number of fragile bits present.

While Hollingsworth *et al* were the first to publish this idea, a note in their paper indicates that Daugman believes that this technique is already being used in commercial iris recognition applications and Masek used this in his open-source code [31].

Once the unwrapped iris has been encoded, the next stage is matching the newly encoded iris to one that was previously stored, and this is discussed in the next section

2.6 Matching

For the matching stage of iris recognition, Daugman [76] and others [60][38][3][30][31][64], create a binary template, consisting of two-bits per pixel, which is called the iris code, as described in the previous section on encoding. This iris code is then compared to the codes in the database using the Hamming distance [100] to calculate dissimilarity scores.

These templates vary in size between researchers, but converting the biometric iris features to binary allows the codes to be compared very rapidly on modern computers and is a very compact way of storing data. The Boolean operator exclusive-OR (\oplus) is typically used to carry out comparisons between the feature vectors of biometric templates. In Daugman's early work [2, 50] he presents the Hamming distance as per equation 2.5.

$$HD = \frac{1}{N} \sum_{j=1}^N CodeA_j \oplus CodeB_j \quad 2.5$$

Equation 2.5 is a summation of the differences between the bits of the two iris codes, $CodeA_j$ and $CodeB_j$, divided by the total number of bits N , in the iris code, for Daugman's method the total is 2048 bits, as this is the number of bits that will fit on the magnetic strip of a credit card. $CodeA$ is the iris code being tested, while $CodeB$ is the code that $CodeA$ is being compared to, and j is the bit number being tested, ranging between one and the total number of bits, N , 2048 in this instance.

By 2001 [81] Daugman had added a mask for each iris code. The mask contains the same number of bits as the code, and is used to indicate which bits of the code are noise, i.e. non-iris data such as eyelashes, eyelids and specular reflections. Daugman sets the bits of the mask that indicate noise to binary zero and the bits that are not noise to binary one.

These masks add two important improvements to the matching process. The first improvement is that noise in the normalised iris can be specifically excluded from the matching process. So, rather than just replacing the noise with an average value, any differences between the two iris codes in locations where at least one of the templates has noise can be removed from the difference calculation.

To compensate for rotational differences between eye images that can be caused, for example, by the eye rotating slightly in the eye-socket or the camera and eye rotating slightly in relation to each other, one of the iris codes is shifted with respect to the other during the matching process. When the iris code is rotated the intersection of iris code and noise bits changes.

By having a separate mask, it can also be rotated so the location of the noise data is also updated. Figure 2.16 to Figure 2.21 show how this works. Figure 2.16 shows two iris codes with the pairs of bits grouped together and the noise pairs marked in pale orange for clarity. It will be remembered from the previous section on encoding that the iris encoding process produces two bits for every pixel of the image.

Code 1	11	00	10	01
Code 2	10	01	00	10
XOR \oplus	01	01	10	11
HD	$5/8 = 0.625$			

Figure 2.16 – Example iris codes with XOR calculation and Hamming distance calculation

The XOR line in Figure 2.16 shows the result of a bitwise exclusive-OR operation carried out on the two codes. The HD line shows the hamming distance calculation and result. No mask is used at this point but the noise has been set to '00'. The Hamming distance (HD) returns a high dissimilarity score of 0.625.

Despite the use of bit pairs for marking the phase quadrature output from the encoding process, the Hamming distance ignores the bit pairs and compares individual bits. Thus, even though in binary the result would be $1_{10} + 1_{10} + 2_{10} + 3_{10}$ making seven out of a possible 12, the Hamming distance simply counts the number of bits at logical one and divides it by the number of bits that could be logical one. Figure 2.17 shows the results after one rotation, the Hamming distance has fallen, but is still showing a high dissimilarity.

Code 1	11	00	10	01
Code 2	01	00	10	10
XOR \oplus	10	00	00	11
HD	$3/8 = 0.375$			

Figure 2.17 – Example iris codes with XOR calculation and Hamming distance calculation.

Code 2 has been shifted once to the left

Figure 2.18 shows the results from the final rotation, again the Hamming distance is 0.375.

Code 1	11	00	10	01
Code 2	00	10	10	01
XOR \oplus	11	10	00	00
HD	$3/8 = 0.375$			

Figure 2.18 – Example iris codes with XOR calculation and Hamming distance calculation.

Code 2 has been shifted left a second time.

Figure 2.19 shows the same two codes in the initial rotational positions shown in Figure 2.16, but this time has added the masks and the bitwise AND operation.

Code 1	11	00	10	01
Code 2	10	01	00	10
XOR \oplus	01	01	10	11
Mask 1	11	00	11	11
Mask 2	11	11	00	11
AND \cap	01	00	00	11
HD	$3/4 = 0.75$			

Figure 2.19 – Example iris codes with XOR calculation, the AND operation using masks and the Hamming distance calculation

The masks in Figure 2.19 show '11' where the iris code locations are valid and '00' where they are noise. The AND operation logically AND's the XOR, Mask 1 and Mask 2 lines. The HD line is the calculation and result of the Hamming distance. In Figure 2.19 the Hamming calculation is $3 \div 4$, or the number of non-noise logic 1's divided by the number of non-noise, or valid, bits.

The Hamming distance in Figure 2.19 shows a very high dissimilarity score, even higher than previously seen. Figure 2.20 shows the results after one rotation, the Hamming distance has fallen, but is still showing a high dissimilarity.

Code 1	11	00	10	01
Code 2	01	00	10	10
XOR \oplus	10	00	00	11
Mask 1	11	00	11	11
Mask 2	11	00	11	11
AND \cap	10	00	00	11
HD	$3/6 = 0.5$			

Figure 2.20 – Example iris codes with XOR calculation, the AND operation using masks and the Hamming distance calculation. Code 2 has been shifted once to the left

Figure 2.21 shows the results from the final rotation. In this instance the Hamming distance is zero. The masks have removed the noise from the Hamming distance calculation

Code 1	11	00	10	01
Code 2	00	10	10	01
XOR \oplus	11	10	00	00
Mask 1	11	00	11	11
Mask 2	00	11	11	11
AND \cap	00	00	00	00
HD	0/4 = 0			

Figure 2.21 – Example iris codes with XOR calculation, the AND operation using masks and the Hamming distance calculation. Code 2 has been shifted left a second time.

The use of masks when using an unwrapped iris code has become very common and the Hamming distance equation in Daugman's later work is adjusted to that shown in equation 2.6.

$$HD = \frac{\|(CodeA \oplus CodeB) \cap MaskA \cap MaskB\|}{\|MaskA \cap MaskB\|} \quad 2.6$$

The dissimilarity between *CodeA* and *CodeB* in equation 2.6 is calculated using the XOR (\oplus) operation, this dissimilarity matrix is then ANDed (\cap) with the two masks and the norm $\|\cdot\|$ taken. This figure is then divided by the norm of the two masks ANDed together. The operations described in equation 2.6 are repeated for each shift of the iris code and its mask.

Very little has changed with regard to the Hamming distance calculation since the addition of masks towards the end of the 1990's. However, in their published works, [101, 102, 28], Rathgeb *et al.* show that the results from the Hamming distance can be improved slightly by looking at the variation of Hamming distances across the shifts and by creating a weighted mask that is updated with each successful authentication. Modest improvements in false accept rate (FAR) of approximately 1% to 1.5% were claimed. This showed that there is still room to improve recognition results at all stages of the process.

Several other methods of improving matching performance when using binary iris codes have been proposed.

Davida *et al* [103] proposed a method based on majority voting with the aim of overcoming the observed bit error rates of 10-20%. By acquiring and encoding a number of iris images during enrolment, each bit in the iris code can be voted on by all the codes collected. So if, for a given location in the template more than half have binary one's then the final iris code will have a binary one in that location. The same applies to a majority of zeros at a particular location. In order to avoid deadlock, where there are an equal number of ones and zeros, an odd number of iris images must be captured and encoded. No experimental results are provided.

Yang and Verbauwhede [104] proposed a method that also acquires multiple images during enrolment but looks for identical code locations. Locations in the iris codes that have the same value (0 or 1) are treated as correct, all other locations in the iris code are treated as noise. Yang claims a reduced false reject rate (FRR) from 6% to 0.8% using this method.

Ziauddin and Dailey [65] proposed a method that weights the bits within the template to indicate which bits are most reliable. Improved recognition results were reported, with the proviso that varying conditions between enrolment and verification time may cause even the stable bits to change.

Hollingsworth *et al* [99] investigated the existence of fragile bits within an iris code and suggested that the central rings of the iris may be more reliable. This is contrary to previous research, such as that carried out by Murugan and Savithiri [33], which suggested the inner parts of the iris were best. Hollingsworth *et al* went on to investigate fragile bits further and discovered that there was a pattern in the location of the fragile bits which could be used within the matching algorithm to improve recognition rates [89].

One problem within feature-based iris recognition is the significant influence that parameters such as spatial location of the iris, orientation and size have on the matching performance. The quantisation that occurs with phase quadrature encoding can require the various parameters to be carefully tuned. Instead of phase quadrature encoding, it has been proposed by several researchers [105, 106, 107, 108, 109] to match using the phase output

of the Gabor filters. One disadvantage of this method, for which Zhang *et al* [109] proposed a solution, is its poor noise immunity.

The improvements in the matching algorithm that are proposed by the researchers mentioned here, all use prior knowledge of the iris codes from a given iris to improve recognition rates. Despite this research, to date, the Hamming distance of phase quadrature iris codes is still the normal way to compare irises.

The work outlined in this thesis does not significantly address the matching or encoding stages of the recognition process. However, the papers discussed in this section significantly increased understanding of the iris recognition process and prompted several proposals for future work.

2.7 Generic Platform

Before work could begin on the generic platform, it was essential to understand the efforts of others in this field. However, while many researchers work in the area of new algorithm development, there seems to be little attempt to provide a generic platform for testing and analysis.

Four open-source and three closed-source iris recognition algorithms were discovered, but none was a development Graphical User Interface (GUI) that aided the improvement and evaluation of iris recognition algorithms.

Masek [31, 29], Li [12, 27], Antonino [38] and Rosa [43, 42] all created MATLAB scripts to perform some or all of the stages of iris recognition but these are not generic platforms. Masek, Li and Antonino all released their code as open-source so anyone can use, modify or re-purpose their code. Rosa charges for his code and uses a hybrid of MATLAB and C code to boost performance.

Two open-source iris recognition algorithms that are not written in MATLAB are JIRRM [40], which is written in Java, and OSIRIS [110, 39] which is written in C/C++ but, again, these are not generic platforms.

GIRIST (Grus IRIS Tool) is a freely available commercial application from GruSoft [44, 111] which is a GUI front end that demonstrates the commercial Giris SDK. However, it is not free and it is not, itself, a generic platform.

Looking at the literature there are many iris recognition algorithms [87, 83, 105, 90, 3, 78, 63, 55, 52, 88] and all authors have developed software to demonstrate their algorithms, but none is a generic platform. Investigations so far have found no evidence that such a platform exists or is being developed.

However, systems that contain partial or complete recognition processes, and have source code available, could be incorporated into the generic platform where their performance could be measured. It was also noted that six out of the nine available algorithms used MATLAB as their language of choice, although finished software will nearly always be developed in “C”, or similar.

Clearly there are many researchers working in the area of iris recognition, and all are expending time and effort creating their own software to demonstrate their own work. Not only is this a waste of resources, but the individual software created means that often the results obtained cannot be compared. There is an urgent need for a single piece of software that can run all these different algorithms, at minimal effort, and in such a manner that results can be compared. This work has produced such a platform.

2.8 Conclusions

This chapter looked at the history of iris recognition, the research carried out within the six stages of iris recognition and the available open-source iris recognition algorithms.

Literature concentrating on all six stages of iris recognition was identified and discussed. Seven works from the literature in particular from, Lin *et al.*, Lu, Gu *et al.*, Shamsi *et al.*, Hollingsworth *et al.*, Aydi *et al.* and Emerich *et al.* were identified for further investigation or comparison.

While this chapter has not explicitly discussed the whole field of iris recognition, it has looked at the main bodies of work in the field. It is clear that they all have advantages and disadvantages, with none being better, overall, than the others. The various open-source algorithms were shown to all be iris recognition algorithms, with none of them providing a truly generic platform that allows the testing and evaluation of any algorithm, already known or yet to be developed.

Such a platform would draw together all previous work, and would be available for any future work, and the development of such a platform is a main objective of this work.

The next chapter describes the various design decisions that were made during the work and the reasoning behind them.

3 Designing the Generic Platform

As stated in the introduction in Chapter 1, the aim of this project was to develop an open-source generic software platform that would facilitate the research and development of new algorithms for iris recognition. Having looked at papers, released code, and the popularity of Masek's code [31, 29], it was clear that there was no generic software platform aimed at aiding the development and evaluation of iris algorithms via a simple GUI.

Given the number of researchers, the variety of results formats and the lack of validation for most of these results, it was clear that a standard development and evaluation platform was needed. In order to meet the industrial requirements to release the generic platform publicly and to be a truly useful tool the generic platform had to meet these criteria:

- It had to include a development environment that promoted fast development and be usable on different operating systems. (Section 3.1)
- It had to be able to output results in a form that would provide consistent, measurable, feedback of algorithm performance, thus, facilitating improved understanding of iris recognition (Section 3.2)
- It had to be possible to load and / or add different iris recognition operations without re-writing or recompiling the platform. (Section 3.3)
- It had to have all the different operations required for iris recognition in-built, but the default operations had to be replaceable with user-defined operations (Section 3.4)
- It had to allow the user defined operations and the in-built operations of the same type to be interchanged quickly and easily to facilitate process evaluation (Section 3.5)
- It had to provide the ability to quickly and easily load multiple image datasets into the GUI (Sections 3.6 and 3.7)

Figure 3.1 summarises the six steps of iris recognition, as discussed in Chapter 2.

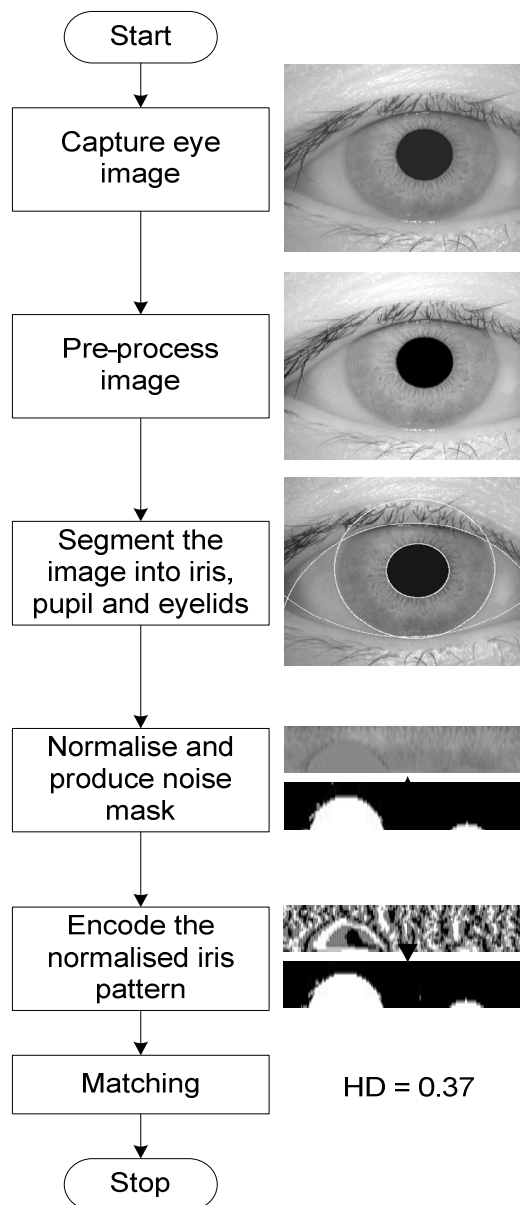


Figure 3.1 – A generic Daugman iris code generation process

Unfortunately, developing capture hardware and/or interfacing to the many capture devices available is an expensive and time consuming task, and would have left little time for algorithm research. Thus, the “Capture eye image” stage in Figure 3.1 was not implemented in the generic platform, instead eye images are loaded from the computers backing store, such as a hard disc, memory stick or network drive.

The pre-processing stage was merged into the segmentation stage because correct segmentation is dependent on optimal pre-processing and there are sometimes multiple loops between these two steps before segmentation is complete. This removed flexibility

that in hindsight would have been useful, but it simplified the interface considerably as the pre-processing is not always carried out at the start of the stage.

This chapter describes the decisions made in the process of designing this platform. The language selection process is discussed in Section 3.1, this is followed by a discussion on measuring the performance of iris recognition algorithms and how it was planned to do this within the generic platform.

Section 3.3 looks at the method used to enable the generic platform to change between different algorithms without recompilation or editing of the core generic platform code. How the generic platform provides the necessary additional algorithms for those developing just one stage of the process is discussed in Section 3.4.

Section 3.5 introduces the methods used to allow the user to quickly switch between algorithms. The method used for adding and quickly switching between image datasets is addressed in Section 3.6.

The iris image datasets available to this research are then discussed in Section 3.7 with reasons given for the selection of the subset used for testing.

The final section looks at how the finished generic platform GUI works.

3.1 Development Environment and Coding Language

This section looks at the selection process used in arriving at a development language and environment for the generic platform [112]. Since the purpose of this section is to show the reason for using MATLAB, only data that was available at the time the decision was made is shown.

3.1.1 Background and Thinning the Field

The purpose of all programming languages is to enable humans to instruct a processing unit to do work. Processing units (processors) have a base instruction set which is tailored to the purpose their designers envisioned for that processing unit. This base instruction set, normally called machine code, is not easily readable by humans, so alternative languages have been devised that are closer to our natural language.

There are many programming languages in the world, from general purpose languages such as assembly language, C/C++ and Pascal to languages designed for specific tasks such as numerical computing (Fortran, Maple, MATLAB) and user application focussed languages like Python, VB, Java and C#.

Of the requirements for the generic platform set out at the start of this chapter, the programming language chosen for the generic platform had to meet these criteria:

- It had to include a development environment that promoted fast development.
- It had to be able to output results in a form that would provide consistent, measurable, feedback of algorithm performance.
- It had to be possible to load and / or add code modules without recompiling the entire platform.
- It had to be usable on different operating systems.

Developing an iris recognition process is a long and complex task. Images have to be captured, processed through to iris code, and then stored in a database. Software to compare newly acquired images to those already in the database must then be written, along with code to analyse the effectiveness of the proposed method.

Writing code can be a long-winded process of modifying, compiling, executing and debugging so a good development platform should include a good environment for these activities.

To provide measurable feedback on the performance of an algorithm the system must be able to store and display results, and measure processing time. These are not features that are easy to implement in all languages.

During the course of this research the main desktop OS's in use were Windows, Mac OS and Linux. In order to appeal to the greatest audience the generic platform should work on as many of these as possible.

Provision of a development environment that promoted fast development implied another criterion, that of using a widely used language. From the list of existing open-source iris recognition algorithms (Section 2.7), the languages in use by other researchers who have published their code were C/C++, Java and MATLAB. All three languages could be used across all of the desired operating systems. This narrowed the field of potential languages significantly.

3.1.2 C/C++

Denis Ritchie created 'C' using a DEC PDP-11 running UNIX in the 1970's [113]. Ritchie and Brian Kernighan released the first book on the 'C' language in 1978. Ratification of the 'C' programming language as an ANSI standard occurred in 1989, with ANSI C compilers becoming available in 1990.

In 1980 Bjarne Stroustrup of Bell Laboratories created his extension to 'C' which he called *C with classes*. In 1983, *C with classes* was renamed to C++. The purpose of C++ was to allow better modularisation of ever-increasing volumes of code, allowing better management of these large programs. The first ANSI standard for C++ was released in 1989.

C and C++ are known as 'mid-level' languages as they combine the power of assembler and machine code with the advantages of portability and well defined structure that comes with a high level language.

At the start of this project C and C++, between them, made up nearly 28% of the Tiobe ratings [114]. They combine portability, flexibility and performance with the ability to easily include assembly language, if access to more specialised features of a given processor is required to achieve even higher performance. Since Linux, Windows and Mac OS on desktop machines now typically run on Intel processors this would not considerably reduce the portability of any program (if care was taken).

Although C/C++ are by their nature cross platform, when applications are written for a particular OS they must use the native OS application program interface (API) to do any significant work. The API is the set of functions that the application programmer has access to in order to direct the OS. This native API code is specific to each OS and this stops the program being portable. The normal solution to this is to use a GUI toolkit, which is a generic API layer that translates the requirement of the program into the specific calls of the host OS.

C/C++ would also require the selection or creation of image processing libraries and, again, these would need to be cross-platform.

3.1.3 Java

Designed in the early 1990's by Sun Microsystems to be a platform independent language for consumer electronic devices, the first version of Java was released in 1995 [115]. Around the same time, another new technology was gaining popularity, the internet.

The internet is made up of many different machines with different CPU's and different operating systems (OS's), yet their users all expect to be able to access the same content and run the same applications. Java fitted this purpose admirably, and this was soon realised by the team at Sun.

Java was designed to be very similar to C/C++, with its syntax inherited from C and its object model adapted from C++. This makes moving between the two languages very straightforward once one of them is learned. C++ and Java code are not compatible, but moving code between C++ and Java is much simpler than between other languages.

Being cross-platform by design, Java meets one of the requirements immediately. Several compilers and IDE's are available, notably Eclipse (<http://www.eclipse.org/>), which is open-source and free to download, making Java very accessible.

When it was first released Java code was considerably slower and more resource-hungry than the equivalent C/C++ code. However, over the years Java has been significantly optimised, leading to much discussion about its performance when compared to C [116, 117, 118].

There is no doubt that Java's performance with respect to C/C++ has improved significantly, but, as was the case during the assembler-versus-C/C++ discussions in the 1990's, the performance comparison is far from clear, coming down to a combination of the particular task and the quality of the implementation.

For this project, Java's advantages would be its popularity, simplified memory management and cross-platform nature. However, Java suffers from the same task-specific disadvantages as C/C++ in that it requires an additional image processing library to be selected or written, as well as being potentially slower than C/C++.

3.1.4 MATLAB

In the late 1970's, Cleve Moler created MATLAB using FORTRAN. His idea was to enable his students to use the EISPACK and LINPACK mathematical libraries without writing FORTRAN code. This was re-coded into C in the early 1980's by Jack Little and Steve Bangert who, together with Moler, formed MathWorks in 1984 [119].

MATLAB is a numerical computing environment for solving mathematical challenges. It has control flow statements, functions, I/O, data-structures and object oriented programming features. Being primarily an interpreted language, MATLAB is able to offer the flexibility of short 'instant' code execution of code typed at the command line, right the way up to fully fledged GUI applications using its built in GUIDE GUI Development Environment.

MATLAB is available for Windows, Linux and Mac OS and no code re-compilation is required to run scripts written on any of the supported OS's. It also comes with an image

processing toolkit that offers many of the functions required for iris recognition, such as edge detection, loading and saving images, convolution and palette manipulation functions.

One of MATLAB's real strengths is its development environment, which provides a neat development cycle allowing a main script to continue 'running' while dependant scripts are modified. The modified dependant script can then be executed from the still running main script. This is not unique to MATLAB, as most scripting languages allow this, but it is unique within the scope of suitable languages for this project.

One area of contention for MATLAB is performance. While most of its individual functions are very highly optimised and return results in very short timeframes, the scripting language can be very slow.

As with all languages, the efficiency of the final code varies depending on the implementation. Even though MATLAB does not work at real-time speeds, it can be used to compare the performance of one algorithm against another, assuming they are both run under the same MATLAB environment.

However, MATLAB is a proprietary product, which subjects users to vendor lock-in and significant cost of purchase and maintenance. It also has a mixed heritage leading to atypical syntax, e.g., the use of curved parentheses '(' and ') for both array indexing and function calling, and lacks support for references. This lack of reference support makes modern data structures with indirection, such as linked lists and open hash tables, difficult to implement.

3.1.5 FreeMAT

Samit Basu created FreeMAT with the aim of replicating MATLABs features in an open-source package [120]. Overall, this aim has been achieved with a high degree of compatibility with the MATLAB commands achieved.

FreeMAT runs natively on Windows, Linux and Mac OS and is driven using a native graphical user interface. The one feature that FreeMAT lacks, however, is the ability to create GUI driven scripts. This is important to this project as the generic platform was intended to be a GUI based tool.

3.1.6 Octave

John Eaton started development on Octave in 1988 and it is named after Octave Levenspiel, an engineering professor at Oregon State University [120]. Like FreeMAT, Octave aims to be an open-source alternative to MATLAB and achieves a high degree of MATLAB command compatibility.

Octave is command line only and runs natively on Linux and Mac OS. Cygwin and MinGW versions are available for those wishing to use the tool on Windows. In 2013 work started on a GUI for Octave and at the time of writing this is in beta testing. There is also now work on a MATLAB compatible GUI, but this is still in alpha testing [121]. Neither of these GUI features were available in 2010 when the language decision was made.

3.1.7 SciLab

SciLab was started by the French National Research Institution (INRIA) in 1990, again aiming to provide a free alternative to MATLAB [120]. Its code is not as compatible with MATLAB as is that from FreeMAT or Octave, but a converter is provided to minimise the impact of any differences.

SciLab runs natively on Windows, Linux and Mac OS and uses a very fast native GUI interface. SciLab also includes the ability to create GUI's for driving scripts, though the GUI components (buttons, menus etc.) must be created using commands so there is no what-you-see-is-what-you-get (WYSIWYG) GUI editor included as standard.

There are two tools available, called ATOMS, which simplify the GUI creation process. The first, called GUIMaker, reduces the code required to generate the GUI to a single line per GUI element. The second, called GUI Builder, presents a WYSIWYG GUI editor but it is very primitive lacking basic element drop-and-drag capabilities.

3.1.8 Conclusion

C/C++ was the preferred choice for this work as it can yield very fast code, is cross platform and there are plenty of free integrated development environments (IDE's) available such as Eclipse [122], CodeBlocks [123] and CodeLite [124]. There are also GUI toolkits such as wxWidgets [125], and image manipulation libraries like OpenCV [126] that are available to help simplify the development process. Most commercial applications of iris recognition also seem to use C/C++, so it is a popular choice.

An initial investigation into developing the GUI in C/C++ was carried out, looking at GUI toolkits and image manipulation libraries. Creating a cross platform GUI is relatively straightforward using a GUI toolkit such as wxWidgets but the code must be re-compiled for each OS. This means that computers with Windows, Mac OS and Linux as well as all the compilers and libraries for each OS were needed for development and testing. The module system for Windows would also have been very different from that on Mac OS and Linux.

Most C/C++ image libraries are aimed at providing the fundamental building blocks for image manipulation and coding complex functions can still be very time-consuming. The debugging features in C/C++ are also lacking in finesse, the most obvious shortfall being the inability to look easily at the contents of numerical arrays in many compilers, e.g. visual studio, due to them being treated as arrays of chars and thus the first zero being taken as the end of the data.

All of these problems can be overcome but it was determined that the time required to implement the desired features using C/C++ would be likely to take most of the time available, leaving little time for research.

Although Java simplifies the coding process when compared to C/C++, it still needs a separate library to implement the image manipulation and it still has shortfalls in the area of debugging that increase the debugging time when compared to MATLAB. So again, it was believed that implementation of the desired features using Java would take too long.

MATLAB is proprietary, expensive, slow and the GUI toolkit has limited features and the resulting GUI's are ugly. However, it also has one of the best debuggers available, is cross platform without re-compilation or re-coding and has a very comprehensive library of

image manipulation functions. MATLAB even allows for compilation to p-code, allowing people to release their algorithms for evaluation without providing the source code. The free alternatives to MATLAB are very accomplished but only SciLab has the ability to create any sort of GUI for the scripts. The manual nature of SciLabs GUI creation meant that it was considered too slow to develop with. However, future conversion to SciLab would be a straightforward process, once the GUI was defined.

For any final iris recognition product, it is highly probable that either C/C++ or Java would be the language of choice. However, this research was not to write the code that would execute fastest, but rather to produce a generic platform that would speed up the development and evaluation of iris recognition algorithms.

All the languages considered would have allowed the creation of a generic platform, but the ease of programming, availability of pre-made toolboxes and default cross-platform execution offered by MATLAB, or the MATLAB clones combined with the time restrictions of an EngD made MATLAB, or one of its clones, the right choice for this development.

Of the MATLAB group, only MATLAB itself had all the features required to ease the work undertaken here. However, now that development of the generic platform is complete, and the MATLAB clones have advanced further, it may well be possible to significantly reduce the cost of the platform by migrating it to a MATLAB clone.

That said, and in hindsight, confining the languages/system examined to just those used by other open-source iris recognition algorithms was overly restrictive.

Python is one popular alternative [114] that could have merited consideration. Guido van Rossum started Python at the end of 1989 and released it via USENET in 1991 [127]. Python is a general-purpose programming language used for everything from simple scripts that perform simple tasks, through glue scripts that link complex tasks together, to full-blown compiled programs [128].

Using add-ons such as wxWidgets [125], numpy [129], scipy[130] and matplotlib [131] all of the language features of MATLAB can be added to Python [128] and adding an IDE such as PyCharm [132] or WingIDE [133] completes the functionality. Had this

combination been considered at the time it is possible that the generic platform would have been written in Python instead of MATLAB.

However, given that the aim of the project was to write a very complex system, use it to evaluate the performance of different algorithms, and then develop new better, algorithms, all in a fairly short timescale, making use of available, professionally produced and maintained software like MATLAB may still have been the best overall decision.

3.2 Measurable Algorithm Performance via Results

It is important for an iris recognition system to be able to distinctly separate the inter-class (Same eye) and intra-class (different eyes) results. For the system to be effective it must be able to distinguish between these two classes of results. The generic platform must be able to show how well a particular algorithm performs in separating intra-class and inter-class results, and it must provide sufficient information to enable useful evaluation.

This section looks at the styles of results presented by other researchers and the benefits of these methods. The purpose of this section is to demonstrate the results output used by other researchers, and to that end, many of the graphs are taken from the referenced literature.

The approach that was decided upon for the common platform is discussed in the conclusion.

3.2.1 Genuine versus Imposter Hamming Distance Graph

In his 1992 paper, Daugman [1] talks about the four outcomes being Authentic Accept (AA), Imposter Accept (IA), Authentic Reject (AR) and Imposter Reject (IR). Since authentic individuals should be accepted and imposters should be rejected, AA and IR are the 'correct' responses. Authentic individuals should not be rejected and imposters should not be accepted, making IA and AR the 'incorrect' responses. For any security system, the aim is to maximise the 'correct' responses and minimise the 'incorrect' responses.

If the hamming distances from AA are placed on a graph of Hamming distance against count and then overlaid with the same plot of IR, the area of the graph that these two sets overlap represents the incorrect responses. If this area of overlap is minimised then the recognition process accuracy will be maximised for a given algorithm. Daugman shows the theory of this clearly in his graph of statistical decision theory Figure 3.2.

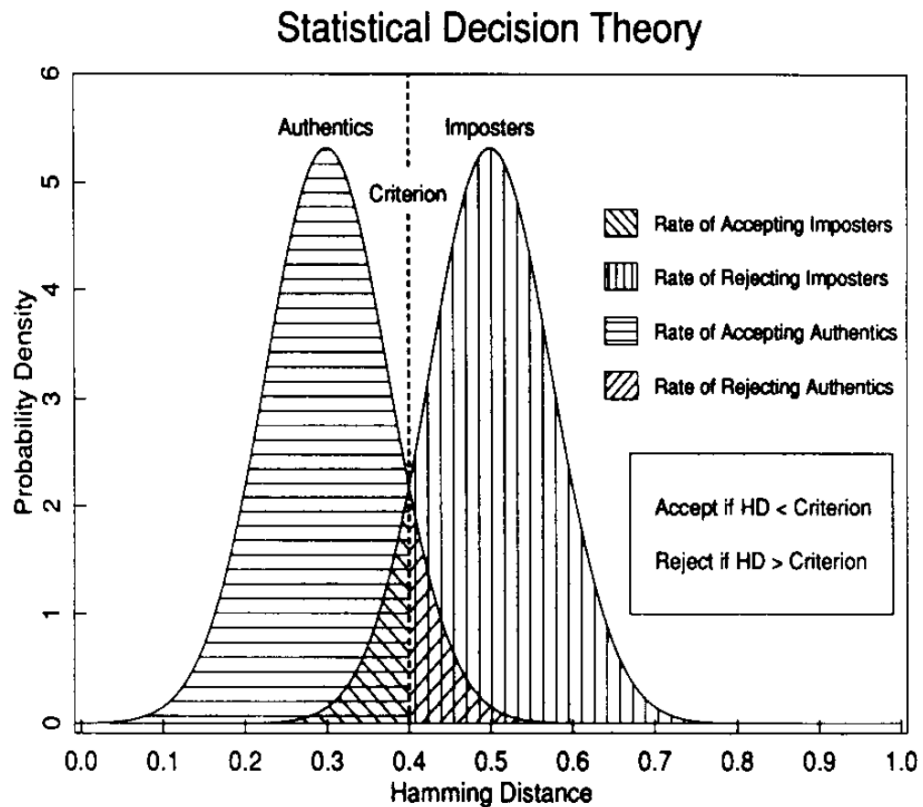


Figure 3.2 – “Statistical Decision Theory: general formulation for decisions under uncertainty” [1]

Setting the Hamming distance threshold to the point marked “criterion” on Figure 3.2 will give the best level of IA and AR, minimising the probabilities of either.

As a tool for finding the Hamming threshold and for spotting how separated the authentic accept and imposter reject groupings are, this is very useful. It shows the degree of overlap, indicating to the developer when remedial action is required. However, without a full analysis of the actual values obtained, comparing results between different algorithms, or versions of the same algorithm, can be very subjective and sometimes difficult. Unfortunately, these values are often not presented in the literature.

Figure 3.3 shows the Hamming distance distribution for authentications and imposters obtained using Segmentation Three on the CASIA V3-interval image dataset.

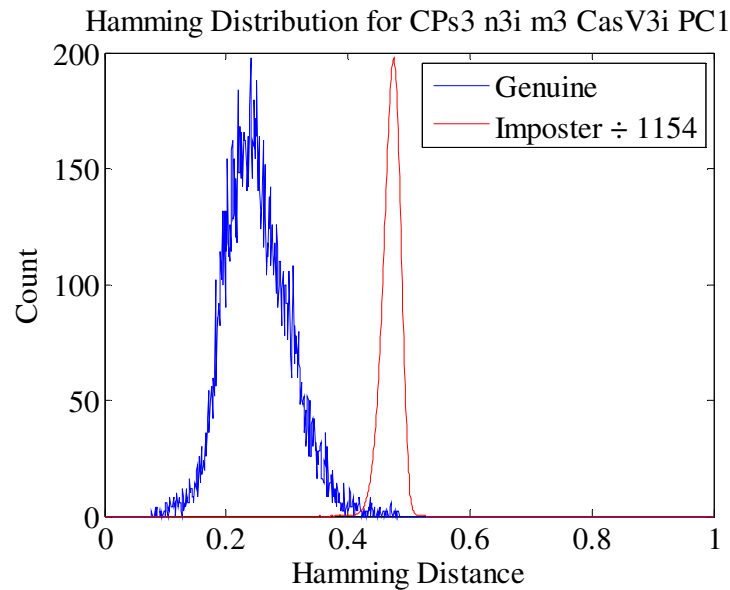


Figure 3.3 – Example Hamming distance distribution for authentications and imposters produced using Segmentation Three on the CASIA V3-interval image dataset. Graph generated using the generic platform.

Figure 3.3 shows a good example of a Hamming distance distribution with the blue authentications Hamming distances clustered evenly around 0.25 and the imposters clustered evenly around 0.45. This shows that using this algorithm the authentications can be clearly identified from the imposters.

The blue authentications and red imposter curves are normalised to compensate for the much higher number of comparisons that are made for imposters when evaluating an algorithm. Without this normalisation the blue authentications curve would be too small to read, as each eye in a dataset will only have between three and twenty images that match, but there will be many more images that do not match.

For example, the CASIA V1 iris image dataset contains 756 images, there are seven images from each of the 108 eyes (classes) present in the dataset. Thus, for each image there should be six authentic images that match and 749 images that are imposters and so do not match.

Figure 3.4 shows an example of a poorly performing algorithm's Hamming distribution.

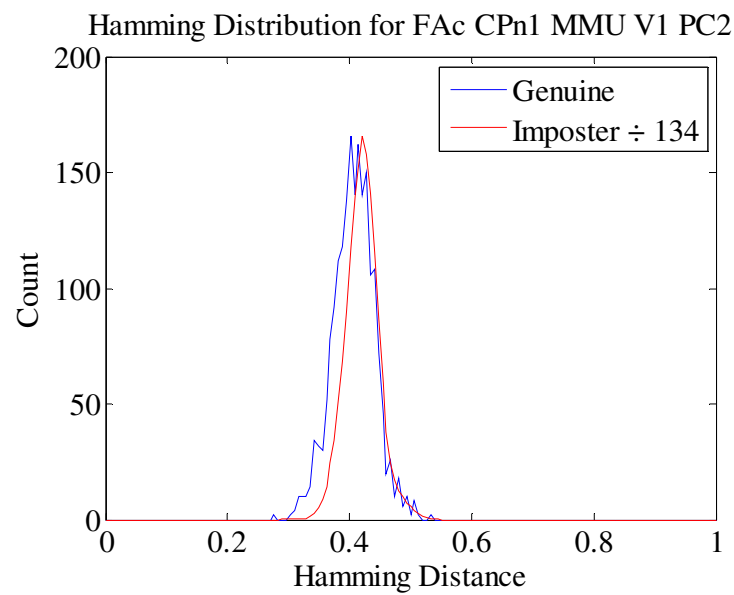


Figure 3.4 – Example Hamming distance distribution for authentic and imposters produced using Antonino's algorithm on the MMU V1 image dataset. Graph generated using the generic platform.

In Figure 3.4 the authentic (blue) and imposter (red) Hamming distance distributions overlap almost completely meaning that it would be very difficult to distinguish authentic from imposters. Thus, Figure 3.4 is an example of what the results from a poor iris recognition algorithm might look like.

3.2.2 Decidability

Daugman proposed using decidability [81], which is similar to the signal detection theory d-prime measure. Daugman defines decidability, d' , as the magnitude of the difference between the intra-class, μ_s , and inter-class, μ_D , means divided by the standard deviation of the intra-class, σ_s , and inter-class, σ_D , distributions. Equation 3.1 shows this mathematically.

$$d' = \frac{|\mu_s - \mu_D|}{\sqrt{\frac{(\sigma_s^2 + \sigma_D^2)}{2}}} \quad 3.1$$

Where d' is the decidability, μ_s is the intra-class mean, μ_D is the inter-class mean, σ_s is the intra-class standard deviation and σ_D is the inter-class standard deviation.

This is far less dependent on the number of comparisons, and less corrupted by outliers than a pure Hamming threshold. Greater separation is indicated by higher decidability figures.

3.2.3 FRR, FAR, EER and ROC

Lin *et al.* [30] used a number of parameters, False Reject Rate (FRR), False Accept Rate (FAR), Receiver Operating Characteristic (ROC) and Equal Error Rate (EER) to demonstrate the effectiveness of their method. The false reject rate (FRR) is the number of irises presented to the system that should be accepted, but are actually rejected. Daugman calls this category authentic reject, AR. The false accept rate (FAR) is the number of irises presented to the system that should be rejected, but are accepted instead. Daugman calls this category imposter accept, IA.

The FRR and FAR are often presented together on a single graph with percentage on the vertical (y) axis and Hamming threshold [100, 76] along the horizontal (x) axis. They are plotted as cumulative percentages at each threshold along the x axis. Figure 3.5 shows an example of a FAR against FRR graph.

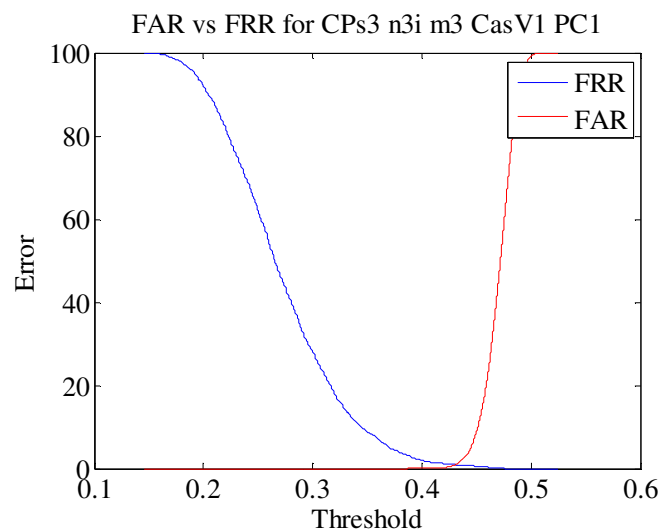


Figure 3.5 – Example FAR against FRR graph for Segment Three with Normalise Three, Masek's encoded and Match Three using the CASIA V1 dataset within the generic platform.

Figure 3.5 shows the FRR in red increasing in percentage from zero at the left to 100 at the right and the FAR decreasing in percentage from 100 percent at the left of the graph to zero percent at the right. This dual graph demonstrates the effectiveness of the algorithm, showing that the FRR does not rise above 0% until a Hamming threshold of 0.38 is reached. The point where FRR = FAR on the two curves indicates the equal error rate (EER), which is 0.9% in this case.

The EER point is also available from the receiver operating characteristic (ROC) curve but is not always visually obvious on the ROC curve and so must be calculated. The EER can be a fast way of comparing the accuracy of different iris recognition algorithms as, typically, a lower EER means higher recognition accuracy.

The ROC graph is used for organising, visualising and selecting binary classifier systems based on how well they perform. ROC graphs are particularly suited to situations where there is a skewed class distribution, as is normally the case with iris recognition. Typically, the *y-axis* is either FRR or 1-FRR. The ROC plot visually characterises the trade-off between the FAR and the FRR. For commercial biometric security systems the matching algorithm performs a decision based on a threshold that determines how close to a template the input data needs to be for it to be considered a match. If the threshold is reduced, there will be fewer false rejects but more false accepts. Thus, a higher threshold will reduce the false accepts but increase the false rejects. Figure 3.6 shows the ROC plot from Lin *et al.*'s segmentation algorithm used with Masek's normalisation, encoding and matching algorithm.

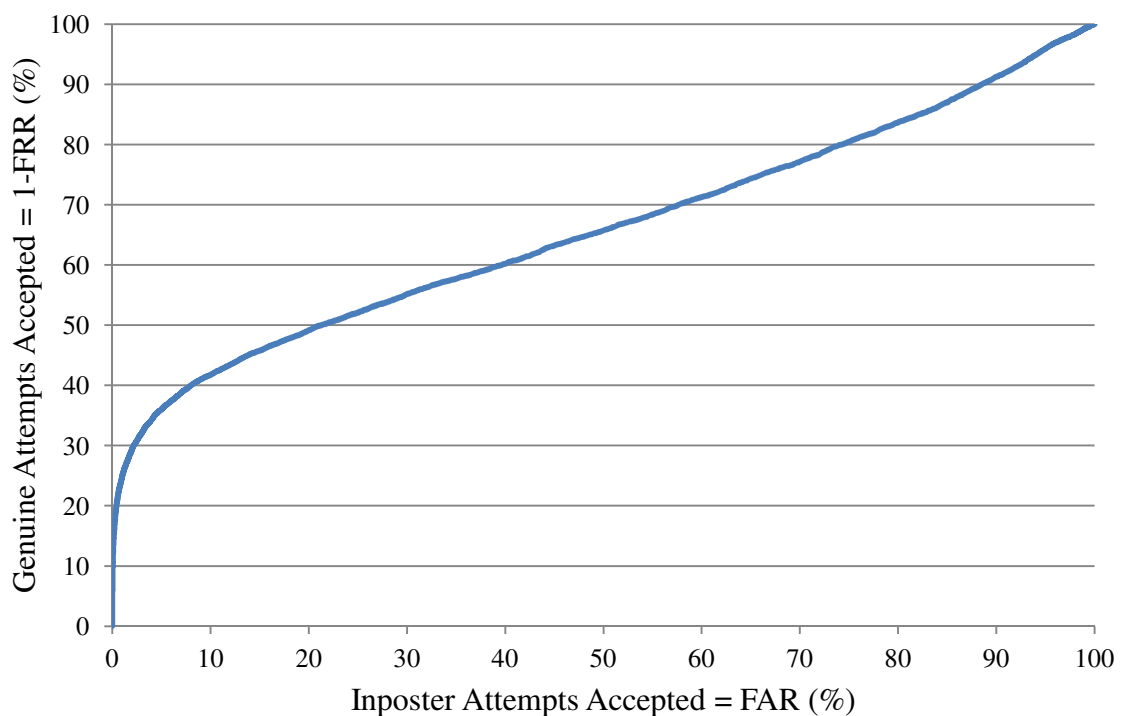


Figure 3.6 – The ROC curve from Lin *et al.*'s method [30]

For a plot of $y=FRR$ the nearer to the origin the curve passes, the better the performance. Thus, Figure 3.6 shows a weak ROC curve as it very quickly moves away from the origin.

Figure 3.7 shows an almost ideal $1 - FRR$ ROC graph produced by the results from Segmentation Three, described in Section 5.3.

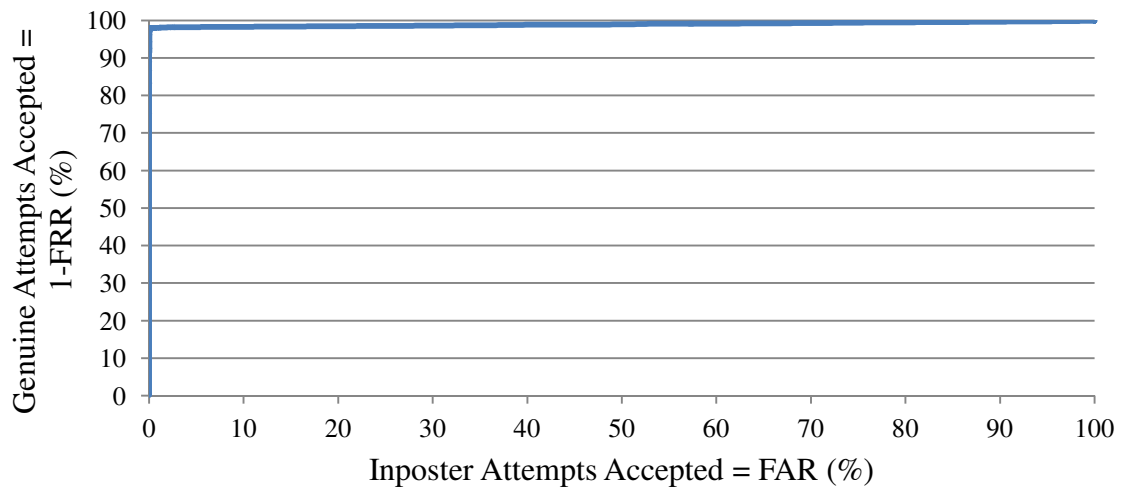


Figure 3.7 – An almost ideal $1-FRR$ ROC graph produced using the generic platform with Segmentation Three using the IITD dataset.

The graph in Figure 3.7 rises vertically up the Genuine Attempts (y) axis to 96.5% then gradually rises to 100% over the remainder of the graph. A perfect algorithm would produce a ROC curve that rose directly to 100% and stayed there.

3.2.4 Conclusion

When looking at the various papers, released code and commercial applications, one aspect that stood out was the lack of a consistently applied test and consistent results format across papers. This is clearly shown in the preceding parts of this section and often makes comparing iris recognition methods quite difficult.

In iris recognition systems designed for security applications, when comparing a presented iris with the enrolled irises, the lowest Hamming distance returned is then compared to a predefined threshold. Hamming distances below the threshold are accepted and those above the threshold are rejected.

The threshold used by security systems is a value that is arrived at by evaluation of the performance of the recognition algorithm and camera capture system being used. The generic platform is used to develop and evaluate recognition algorithms and could not have a pre-defined threshold, as one would not be available. This also removes the ability to improve the results by simply changing the Hamming threshold used.

Assuming that the image dataset provider has correctly named each iris image file, the generic platform is capable of checking that two images are actually from the same eye. This gives algorithms in development an advantage over normal security systems when using the available iris image datasets as the images can be deterministically matched using their file names and / or directory path.

Thus, it is not only possible to characterise these algorithms performance in terms of Hamming distance, but also to provide a simple set of results indicating what percentage of 'best' matches were a genuine iris match (true match) and what percentage were from an unrelated iris (false match).

Returning to Daugman's categories (AA, IA, AR and IR) a true match would be an authentic accept and a false match would be an imposter accept. However, true match and false match go beyond these definitions, as they are actually independently verified results. When Daugman discusses an authentic accept, a great deal of additional work is needed to verify that an "accept" is genuinely authentic.

These figures of true and false match give a very real indication as to whether the systems best match is likely to find an authentic or an imposter. Hence, the additional terminologies of true and false match.

If the Hamming distance groupings of the true and false matches are compared, a third figure, separability, can be derived. This is a numerical representation of the information provided by Daugman's "Hamming distance distribution for authentic and imposters" (Figure 3.2, Figure 3.3).

The separability of the results from a given algorithm is the percentage of true matches for which the Hamming distance is less than the lowest Hamming distance returned by a false match. For the perfect algorithm, there would be no overlap between the true matches and false matches, so the separability would be 100%. Separability is calculated using equation 3.2.

$$\text{Separability} = \frac{\sum(\text{True Matches} < \min(\text{false match}))}{\sum(\text{True Matches})} \times 100 \quad 3.2$$

The decidability of the results from a recognition algorithm is the magnitude of the difference of the average hamming distances from the intra- and inter-class results, over the standard deviation formed from the sums of the variances of the intra- and inter-class results, Equation 3.1. The number that is returned gives an indication of how well the intra- and inter-class results are grouped.

Decidability and separability are very similar metrics, but they do give a different view of the same thing. The separability score can be significantly affected by a single low false match result. One false match Hamming distance significantly lower than the rest will reduce the separability. On the other hand decidability only indicates how well each class is grouped and gives no indication of the extent of outliers.

Together separability and decidability give a much more complete picture of the quality of the results achieved. If both separability and decidability are high then the results from an algorithm are well grouped within each class and setting a threshold will be straightforward.

Thus, the output would contain three graphs (ROC, FAR against FRR and Hamming distance distribution) and a text file summary containing the system info, date, time, experiment configuration and results. The results would include:

- The true and false match rates
- The separability
- The decidability
- The equal error rate (EER)
- The Hamming threshold at which the equal error rate occurs

These results ensure that the generic platform provides sufficient information such that an appropriate Hamming distance threshold can be selected, as well as showing the effectiveness of the algorithm being tested when compared to other algorithms, both potential and pre-existing.

The next section discusses the method used for switching between the loaded iris recognition operations without recompiling the generic platform.

3.3 Changing Iris Operations without Recompilation

Loading and adding different iris recognition operations without re-writing or recompiling the platform required the use of a loadable module system. In MATLAB this would be defined as executing scripts whose name is not known at the time the program is being designed.

MATLAB currently has two methods of executing scripts whose name is not known at design time. The first method is a function called `feval`:

```
[Result1, ..., Result n] =  
    feval('function name', Parameter 1, ..., Parameter  
n)
```

The `feval` function in this form will take a function name string and try to find and then execute that function. So long as the function is stored on the defined or current MATLAB paths, it will be found and executed.

The second method is function handles. These use a variable structure to store all the information that is required to execute a function. The main advantage of function handles is that once they are created the script file location is no longer needed on the path.

Up until MATLAB version seven `feval` was the only way of running scripts whose name is not known at design time. To maximise backwards compatibility the method used in the generic platform is the `feval` command, though, in hindsight, the use of function handles may have produced a more elegant solution, no features were compromised for using `feval` versus function handles.

Requiring the code for operations to be written such that it can be loadable adds a small amount of code overhead, but, once added this overhead does not add any additional processing during encoding or matching and allows the main GUI to be written such that no code changes are needed to make the swaps.

The generic platform calls these specially written scripts, modules. Modules are at the heart of the generic platform, providing the ability to add algorithms and flows easily and

without restriction. Each module contains functions that add either recognition algorithms or supporting functions to the generic platform. The base structure of each function is defined by its inputs and outputs, thus standardising the operation.

Modules are added by creating a folder in the host operating system (OS) file system in the root of the generic platform program folder. The module directories are named consistently starting with the four characters “mod_”, followed by the name of the mod, typically the name of the module’s author. Thus, adding a new module is a simple process for those with familiarity with the host OS file system.

The next section discusses how the requirement for in-built iris recognition operations was met.

3.4 In-built Iris Recognition Operations

The fastest, simplest way to provide all the different operations required for iris recognition built-in to the generic platform was to build existing code into a set of modules to be loaded by the generic platform. This would maintain a consistent interface and meant that changing or updating this was very simple.

Libor Masek produced the first widely known open-source iris recognition algorithm [31, 29] in 2003 using MATLAB. Masek's code demonstrates the principles of iris recognition and contains all the stages of iris recognition except image capture.

The system consists of a set of MATLAB scripts with two top-level scripts, *createiristemplate* and *gethammingdistance*. The first of these, *createiristemplate*, takes a greyscale image, e.g. Figure 3.8, as its input and outputs a binary biometric template, e.g. Figure 3.9.

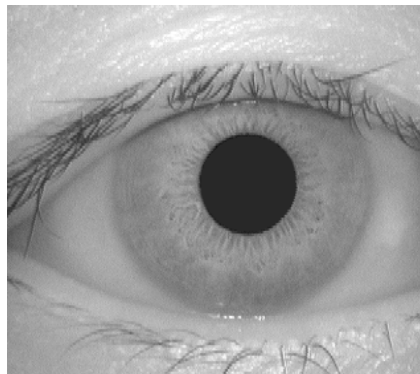


Figure 3.8 – Input image example from CASIA V1 database



Figure 3.9 – Binary biometric template, code (top) with mask (bottom)

Masek's second script, *gethammingdistance*, takes two binary biometric templates with their associated masks, like the one shown in Figure 3.9, and calculates the hamming distance between them.

Although his work was never published in a peer reviewed journal or conference proceedings, Masek's algorithm is very widely referenced in the literature, [32, 33, 34, 35, 36, 37, 87, 88, 86, 89]. His code was also found to be stable and robust making it an ideal algorithm to provide a complete default set of the operations required for iris recognition. Other options such as the code from Li [12, 27] and Antonino [38] were not as widely referenced.

By refactoring Masek's code to be a loadable module for the platform it was possible to make these default operations replaceable with user-defined operations in a consistent manner. Thus Masek's algorithms are just another module, but it is a complete set capable of all the stages of iris recognition without any additional coding by the user.

3.5 Fast Iris Recognition Algorithm Swapping

Allowing the user-defined and in-built operations to be interchanged quickly and easily was achieved using a combination of the module system and a user friendly GUI design. In the code, the module system allows the platform to run any one of the loaded modules for a given recognition operation.

In the GUI, each recognition operation has a listbox that lists all the algorithms loaded for that operation. The user selects the desired algorithm for each operation to be performed from the operations listbox using the mouse.

The next section discusses how image datasets are added to the generic platform.

3.6 Image Datasets

Image datasets are a core resource for the analysis of any iris recognition algorithm, as iris images of many different types are necessary to test the algorithm adequately. Quickly and easily loading multiple image datasets into the GUI required a repeatable methodology that captured as much information regarding the images as possible.

Given the size of many of the image datasets [15, 13, 9, 10, 4, 5, 6, 7], no purely manual method of adding an iris dataset could be considered fast. Thus, the addition of image datasets to the generic platform had to be at least a partially automated process.

A method was proposed that used the operating system's filesystem to organise and store the images in each dataset. Most of the available image datasets are supplied in a compressed archive file that is extracted to a filesystem directory before being used. The user would be asked to specify a root directory in the filesystem to use and then each image dataset would be stored in its own directory inside the root folder. On start-up the generic platform would read the folders in the root image dataset directory and display their names in a list box in the GUI.

This method was implemented and tested within the generic platform. Although it does require a basic knowledge of the creation of directories and file copying within the host operating system these skills are quickly learned and not uncommon. For those with the skills to manipulate the host filesystem, choosing a root directory and adding new subdirectories is a fast and simple task as is extracting the image files to the subdirectory.

During the first run of the platform the user is asked to specify the root folders for the image datasets and the results database, and these settings are then saved to a file in the main program directory as "<HOSTNAME>.opt". Figure 3.10 shows the dialog box for the image database directory.

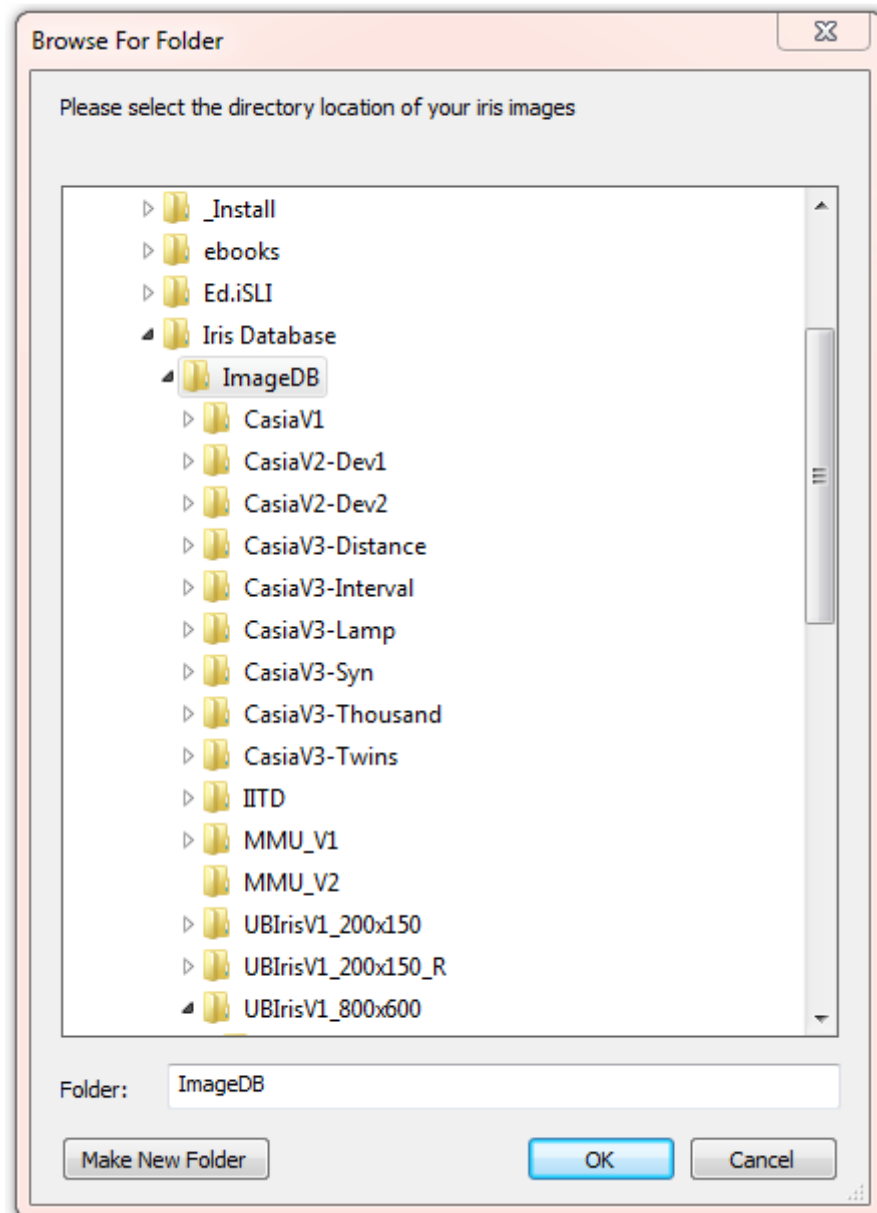


Figure 3.10 – Iris GUI requesting the image database root directory

The name of the image dataset directories in the root image database directory is the name displayed for the image datasets in the GUI, as shown in Figure 3.11.

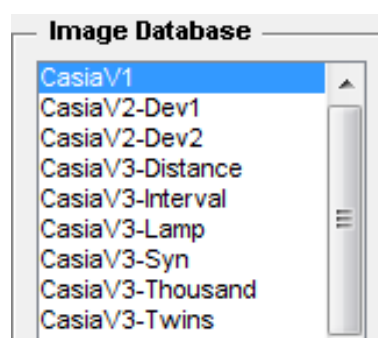


Figure 3.11 – Iris GUI image dataset list

This process takes a few minutes, depending on the number of images in the image dataset being added. When compared to a purely manual process of adding each image file to the code, the method implemented was considered to be fast and easy and fully repeatable.

One concern regarding the automatic method was the lack of any data richness. The most important piece of information used by every researcher is class [83, 76, 99, 30, 31]. In the context of iris recognition the class is the group of images which are pictures of the same eye. Knowing if two eye images are from the same class is the most important piece of information that the generic platform needs, to be able to determine if a match that has been made is between images of the same eye. The entire true/false match system is based on this information.

If the platform cannot determine whether images are from the same class or not then it cannot determine if a best match is of the same class and so cannot calculate the true and false match values. Looking at most of the available eye image datasets, class can be normally clearly determined from either the file name, or a combination of the filename and the path.

This information is used by a script that is added to the generic platform for each image dataset in use. This script looks at the naming convention of the particular image dataset and determines if the two images are from the same class. This script will be relatively short, less than 20 lines of code, unless the naming convention is particularly obtuse. The scripts for all the image datasets available to this research have already been written and are included with the generic platform.

Since the automated method allows for many datasets to be added, should useful information for a particular dataset be available it would be possible to split a single dataset into several datasets along data demarcation boundaries such as age, sex, ethnicity, allowing more detailed analysis of performance to be carried out.

The next section takes a deeper look at the image datasets available and discusses the selection process for the datasets used by this research.

3.7 Image Dataset Selection

As discussed in the literature review, there are a number of image datasets available to researchers in the field of biometrics. This section looks at these datasets, highlights the ones that were used and describes why they were used. The following subsections look at each of these datasets in more detail and examine their suitability for use.

Sections 3.7.1 to 3.7.10 describe each of the individual image datasets available to this research. Section 3.7.11 provides a summary of the datasets, their statistics and features. The final Section, 3.7.12, describes the selection of the datasets used for testing

3.7.1 CASIA V1 [4]

The Chinese Academy of Sciences' Institute of Automation (CASIA) datasets have been referenced in many publications throughout the world [30, 28, 112, 55, 59, 60, 61, 63, 35, 31], [38, 79, 82, 134, 135, 101, 102, 106, 107, 108], [109, 43, 42, 40, 87, 34, 136, 86, 89]. The datasets contain thousands of, mostly Chinese Asian, eye images for general research purposes. Each image is an 8-bit grey-scale image ranging in size from 320×280 pixels up to 640×480 pixels.

CASIA has released four groups of datasets to date [4, 5, 6, 7]. The original CASIA V1 contained a single dataset of 108 individuals with seven images of each eye collected during two sessions. The images are all 320×280 pixels grey scale and were captured with CASIA's proprietary infrared close-up iris camera.

In order to protect their intellectual property, CASIA manually modified all of the pupil's in the images, overwriting them with solid black disc's, thus obscuring the reflections of the lighting configuration used. Unfortunately, CASIA did not include this information with the dataset and until Phillips exposed the datasets modifications in 2007 [137] researchers assumed that captured pupils were solid discs of black [31, 82, 134, 135, 43, 34, 136].

These researchers concluded that the large peak in histogram values below intensity 100 present in the CASIA V1 dataset images was normal and could be relied on as a method

for determining the boundary between the pupil and iris. Histogram thresholding is discussed in more detail in Appendix Appendix A. The dataset continued to be used by researchers as their sole test dataset until 2011 [30, 59, 60, 61, 64, 65, 83, 38, 108, 42] when CASIA added a note to the dataset explaining that the images were modified [4].

The CASIA V1 dataset is still used but only in conjunction with other datasets as any segmentation algorithm results obtained from testing only with this dataset must be treated as suspect, due to the much simpler pupil boundary detection. The dataset is perfectly adequate for normalisation, encoding and matching, but most now use the CASIA V3 Iris-Interval dataset instead.

3.7.2 CASIA V2 [5]

The CASIA V2 iris image library was first used for the Biometrics Verification Competition (BVC) in 2004. There are two datasets containing 1,200 images each. The images are higher resolution than the V1 or V3i sets and contain more shadows with an increased distance from the camera to the subject.

The CASIAv2 database is a well-known publicly available iris dataset of degraded images containing strong eyelid and eyelash occlusions, and varying illumination[106]. Both datasets contain 1,200 images of 60 classes from 30 subjects at 640×480 pixels. The only difference is the device used to capture the images. The CASIA-Iris-Device1 (V2D1) was captured with the OKI IRISPASS-h [53], the CASIA-Iris-Device2 (V2D2) was captured with the CASIA designed IrisCamV2.

3.7.3 CASIA V3 [6]

The CASIA V1 dataset was created from the CASIA-Iris-Interval (V3i) image dataset. The CASIA V3i dataset comprises 2639 images of both eyes from 249 individuals. This is three and a half times as many images as CASIA V1 and none of the images have the pupil and intensity adjustments that marred the V1 dataset.

The CASIA V3i dataset is often used in the literature [101, 102, 28, 63, 14, 79, 87, 86, 89] as its images are all forward looking eyes taken under ideal near infra-red conditions. This

means that the iris textures are distinct and the boundaries clear. There is obstruction from eyelids and eyelashes, and the specular reflections in the pupil from CASIA's lighting arrangement can present a challenge, but these images are among the best quality images available.

Thus, the CASIA V3i image dataset is ideal for early testing of new segmentation algorithms and for working on algorithms in normalisation, encoding or matching due to the images easy segmentation. As the superset of CASIA V1, CASIA V3i is also an ideal dataset for testing iris recognition algorithms that were developed using the CASIA V1 images as it presents them with the best opportunity of succeeding as well as showing if they depend on the CASIA V1 pupil modifications.

The CASIA V3 interval dataset is only one of three datasets that form the CASIA V3 library. The second dataset in the CASIA V3 library is the CASIA-Iris-Lamp (V3L) dataset used by Zhang [109]. The CASIA V3L dataset contains 16,625 images captured by a handheld OKI IRISPASS-h [53] infrared iris camera at 640×480 pixels from 411 subjects giving 819 classes.

A lamp was toggled on and off near the subject so that there was more intra-class image variation. This changing illumination caused the subjects pupils to expand and contract deforming the iris. This elastic deformation of the iris is a very common and challenging issue for iris recognition algorithms to overcome.

The third dataset in the CASIA V3 library is the CASIA-Iris-Twins (V3t). This dataset contains 3,284 images from 200 twins giving 400 classes. These were again captured with the OKI IRISPASS-h, this time outside. This was the first set of iris images of twins made available publicly [6].

The images in the CASIA V3t dataset are of a consistent high quality with much lower variation in illumination and thus elastic deformation of the iris than the CASIA V3L dataset. The use of the OKI IRISPASS-h also means that neither V3L or V3t datasets have the ring of specular reflections on the pupils that are present on the V3i dataset.

3.7.4 CASIA V4 [7]

The CASIA V4 library of iris images contains three new datasets of NIR images. The first new dataset is CASIA-Iris-Distance (V4d). The CASIA V4d iris images were captured with the CASIA developed long-range multi-modal biometric image acquisition and recognition system (LMBS). The LMBS can capture data at up to three metres away using multiple cameras.

The dataset contains 2,567 images in 284 classes from 142 subjects at a size of 2352×1728 pixels. The images are so large because they contain the full face of the subject, this allows multi-modal biometric matching, e.g. face and iris recognition, but makes it difficult to use with recognition algorithms that are only designed to detect a single eye, not the whole face.

The second new dataset in the CASIA V4 library of images is CASIA-Iris-Thousand (V4t). This dataset contains 19,980 images in 2,000 classes taken from 1,000 subjects at a size of 640×480 pixels. The CASIA V4t dataset is the first publicly available iris image dataset from one thousand subjects [7]. These NIR images were captured with an IrisKing IKEMB-100 camera indoors with a lamp being toggled on and off near the subject. The changing environmental lights ensure a good source of intra-class variation in the form of iris elastic deformation as well as the normal occlusion and specular reflections.

The final new image dataset is the CASIA-Iris-Syn (V4s). This dataset contains 9,990 images of 1,000 classes. These images are unusual in that they are not images of real eyes, but are computer synthesised using the CASIA V1 dataset as a base. Tests carried out by Professor Tan and the researchers at CASIA concluded that most of the people involved in the testing could not distinguish between genuine and artificial iris images [7]. They also found that performance results from the CASIA V4s dataset had similar statistical characteristics to tests carried out with genuine iris image datasets.

3.7.5 IIT Delhi [13, 14]

The Indian Institute of Technology (IIT) Delhi iris dataset consists of 1120 iris images from 176 males and 48 females collected at IIT Delhi and was acquired using a JIRIS

JPC1000 [67], digital CMOS camera. The subjects were between the ages of 14 and 55 years. All 224 images are at a resolution of 320×240 pixels in bitmap format.

Like the CASIA V3i dataset, the IIT Delhi dataset contains good quality, well-illuminated indoor images. The interesting difference is that the IIT dataset is mostly Indian shaped eyes, collected from the staff and students at IIT Delhi during the first seven months of 2007.

3.7.6 MMU [8]

There are two iris datasets from the Multimedia University (MMU), each collected using a different camera. The MMU datasets offer a good mix of ethnicity and quality of image to stress the algorithm under test but the image count is low when compared to CASIA.

The MMU V1 dataset contains 450 images at 320×240 pixels with 90 classes taken from 46 subjects. The images were taken using an LG IrisAccess@2200 [54] operating at a range of 7cm to 25cm.

The MMU V2 dataset contains a considerable variety of quality of image, with off-axis, glasses and a variety of lighting conditions. The dataset contains 995 images in 199 classes taken from 100 subjects with varying age and ethnicity. Five left eye images were removed due to the presence of cataracts. These images were taken using a Panasonic BM-ET100US Authenticam [138] at a range of 47cm to 53cm from the subject.

3.7.7 WVU [11, 12]

The West Virginia University (WVU) iris dataset consists of over 865 off-axis iris images, 268 of which were collected with a Sony Cyber Shot DSC F717 [66], while the remaining 597 were collected using a monochrome, black and white camera. There is a slight green hue to the Sony camera images as it was used in infrared mode, but still used all three RGB sensors.

The images are from 100 subjects and 200 classes. The WVU dataset has quite stringent release pre-requisites and is only available on disc, not via download. There is however, a small subset of the WVU dataset of 80 images, identified as ‘WVU Free’ in this text, that is provided without restriction by Li [12, 27] alongside his iris recognition code download.

At only 80 images, this subset of the WVU dataset is very small, but contains images with a greater degree of variation of gaze direction than many others.

3.7.8 UBIRIS [9][10]

The Unconstrained Biometrics: Iris (UBIRIS) V1 dataset contains 1,249 images taken from 241 subjects and classes captured over two sessions at 800×600 pixels. The dataset provides consistent close-up images of eyes provided in two different resolutions as well as colour and greyscale. The main difference when compared to the existing public and free databases at the time of its release in 2004, CASIA and UPOL, is that it incorporates images with several noise factors, thus challenging the robustness of iris recognition methods.

Capture session one focussed on minimising noise factors such as reflections, luminosity and contrast, by installing the image capture equipment in a dark room. The second capture session occurred in a different capture location in order to introduce natural light. This facilitated the appearance of images with diverse reflections, contrast, luminosity and focus problems. These images from the second session simulate image capture by a vision system with minimal or no active subject participation.

The UBIRIS V2 dataset provides a much wider variety of image qualities and distances than other iris image datasets. The dataset contains 11,101 images in 522 classes from 261 subjects at 400×300 pixels.

The UBIRIS V2 database is targeted at research that evaluates the feasibility of visible wavelength iris recognition in non-ideal imaging conditions. The image capture equipment was installed in a lounge under both natural and artificial lighting sources. The subjects were of Latin Caucasian (90%), black (8%) and Asian (2%) ethnicity. Images were

captured while the subjects walked slowly between marks on the floor while looking at lateral marks that caused them to rotate their head and eyes.

3.7.9 UPOL [15, 16]

The University of Palackého and Olomouc (UPOL) dataset contains 384 iris images in 128 classes from 64 subjects. The images are 24-bit, RGB, 576×768 pixels. The irises were scanned using a TOPCON TRC50IA optical device connected to a SONY DXC-950P [69] 3CCD camera. The UPOL dataset is unusual in that each iris is surrounded by a black border, caused by the capture equipment.

This border can make detection problematic for edge based algorithms as there is a very sharp boundary outside the actual iris but the irises are extremely clear with minimal noise from eye lashes and lids. The main source of noise, other than the black border which can be accounted for, is the consistent specular reflection in the pupil from the light source. These two noise sources are very consistent and can be accounted for easily in algorithms; otherwise , these images are very free from noise.

3.7.10 ICE 2005 [17]

The Iris Challenge Evaluation (ICE) 2005 was created by the National Institute of Standards and Technology (NIST) in 2005 as the test dataset for the ICE iris recognition challenge that they organised. The ICE aimed to measure and compare state-of-the-art iris recognition performance on images captured within realistic environments. It was hoped that this would lead to improvements in iris recognition research and development [17].

The ICE 2005 dataset contains 3953 greyscale iris images captured in the near infrared (NIR) by an LG IrisAccess®2200 [54] camera from 132 individuals. The images are of a single eye, with 1425 images of right-eyes from 124 people and 1528 images of left eyes from 120 individuals. This dataset contains a good mix of ideal and non-ideal images including a large number of low-quality iris images which are defocused, occluded with eyelashes and eyelids and deformed [109].

3.7.11 Summary of Datasets

This section summarises the available datasets together with their technical details and features in a tabular form. Table 3.1 lists the available datasets together with their features and technical details. Alternate line shading is used to aid readability.

Table 3.1 – Summary of the image datasets available to this research, the features of each of the datasets and their technical details

Image Datasets										
Dataset	Resolution (X × Y)	Light Wavelength	Image Format	Number of Images	Number of Classes	Total Comparisons	Simplified	Specular reflections	Eye lashes and lids	Obfuscation
CASIA V1	320×280	NIR	BMP	756	108	571k	✓		✓	
CASIA V2D1	640×480	NIR	BMP	1,200	60	1.4M		✓	✓	
CASIA V2D2	640×480	NIR	BMP	1,200	60	1.4M		✓	✓	
CASIA V3i	320×280	NIR	JPEG	2,639	395	6.9M		✓	✓	
CASIA V3L	640×480	NIR	JPEG	16,625	819	276M		✓	✓	
CASIA V3t	640×480	NIR	JPEG	3,284	400	10.8M		✓	✓	
CASIA V4d	2352×1728	NIR	JPEG	2,567	284	6.6M		✓	✓	✓
CASIA V4t	640×480	NIR	JPEG	19,980	2,000	399M		✓	✓	✓
CASIA V4s	640×480	NIR	JPEG	9,990	1,000	100M		✓	✓	
IITD	320×240	NIR	BMP	2,240	224	5M		✓	✓	
MMU V1	320×240	NIR	BMP	450	90	202k		✓	✓	
MMU V2	320×238	NIR	BMP	995	199	989k		✓	✓	✓
UBIRIS V1	800×600	Vis	JPEG	1,249	241	1.6M		✓	✓	
UBIRIS V2	400×300	Vis	TIFF	11,101	522	123M		✓	✓	✓
UPOL	786×576	Vis	PNG	384	128	148k	✓	✓		
ICE 2005	640×480	NIR	BMP	3953	244	15.6M		✓	✓	✓
WVU	640×480	NIR	BMP	865	200	747k		✓	✓	✓
WVU Free	640×480	NIR	BMP	80	20	6k		✓	✓	✓

The dataset column in Table 3.1 is the name used by the generic platform to identify each dataset. The resolution column lists the dimensions of the picture in pixels, the horizontal (X) dimension followed by the vertical (Y) dimension. This size information gives an indication of how detailed the images are and also how long they may take to process, though other factors such as the segmentation algorithm, edge and circle detection methods can have a significant impact on the processing time.

The light wavelength at which the images are captured impacts the recognition process significantly. Most image datasets from the release of the CASIA V1 dataset onwards have used the near-infra-red (NIR) spectrum, as at these wavelengths light passes straight through the eye's melanin pigment to clearly reveal the iris structure [76, 31, 6]. For this reason, NIR images show the details of the iris very clearly, generally improving the amount of detail that can be used for the encoding stage of recognition. NIR image capture is the normal method for capture carried out at less than one metre [78, 19, 18].

At greater distances, image capture in the near-infra-red is more difficult [139]. This difficulty has increased research interest in visible wavelength (Vis) image capture [9, 10, 139, 22, 57, 62, 15, 16] in order to reduce the intrusiveness of iris recognition.

Iris images taken using visible light are typically more difficult to process as the melanin in the iris, especially for darker iris colours, can obscure much of the iris texture detail as well as making the pupil harder to detect. Visible light capture is the target for projects looking at less constrained iris image capture [25, 139], however, only four of the datasets available to this research had images captured using visible light.

The 'Image Format' column indicates if the images are saved in a compressed format like JPEG, TIFF or PNG, or an uncompressed format such as bitmap (BMP). According to the literature [58, 57, 59], even severe lossy compression has minimal impact on recognition performance, but it does have an impact. Therefore, it is important to be aware if the images being used are compressed or not as this can have a small impact on recognition performance.

The next three columns in Table 3.1 indicate the statistical significance of the image dataset for the purposes of testing an iris recognition algorithm. The 'Number of Images' column states the number of individual images in each image dataset. The more images

there are, the more iris codes will be created and the greater the 'Total Comparisons' will be. More comparisons means greater confidence in the results obtained. The 'Number of Classes' column indicates the number of unique eyes for which the dataset contains images. This is not the same as the number of individuals as each individual typically has two eyes that can be captured, so the two eyes from each individual will form two separate classes.

The 'Simplified' column indicates if the images have been simplified in any way that makes them easier to segment. The CASIA V1 dataset was manually modified making the pupil detection easier, UPOL has no eyelid or eyelash noise and has a black border which covers most of the facial noise, again making segmentation easier.

The final three columns in Table 3.1 indicate what sources of noise are typically present in the images of each dataset. 'Specular reflections' on eye images are mirror like reflections of the light sources used during image capture. They can occur in both the pupil and the iris or even across the boundary between the two. Typically any iris data under the reflection is unusable and in the pupil the specular reflections can confuse the segmentation algorithm.

For controlled set-ups such as those used by the CASIA datasets, the specular reflections will be consistent across images. For datasets with more variety of environment and position, the reflections will vary across the images.

'Eye lashes and lids' obscure any boundaries and iris texture that they cover. These noise elements can also move significantly between images of the same eye. Successful detection and masking of these elements will improve the recognition results obtained.

'Obfuscation' is a deliberate act on the part of the subject. Examples of obfuscation within an image can include wearing glasses or contact lenses, keeping the eyelids as closed as possible or simply directing the eyes to focus off to one side, away from the camera.

The more sources of noise that are present in an image, the harder it will be to segment and potentially the smaller the amount of real data available to encode. Algorithms that perform well, even with many sources of noise are the goal of iris recognition research.

3.7.12 Selection of Datasets for Testing

This section discusses the rationale behind the eight iris image datasets selected for use in the testing of the algorithms in this text and the reasons for excluding the rest from testing.

Many researchers focus on just one dataset, two at most, for their research [12, 31, 28, 30, 22, 24, 59]. This is generally helpful for making a specific point in the literature, but it limits the comparative usefulness of the results and generally leads to algorithms that are carefully ‘tuned’ to work with a particular dataset. The goal for iris recognition must surely be the development of algorithms that are as image quality and source agnostic as is possible.

To increase impartiality and diversity of image quality, eye shape and iris colour for testing, this research would use more than two image datasets. For these reasons twelve eye image datasets were added to the repository of images, so allowing testing to be carried out using as great a variety of input data as possible.

However, the datasets listed in Table 3.1 comprise 79,558 images of varying degrees of quality, size and colour depth. It would not be possible to test with all of these for every algorithm and variation so a broad subset of near infrared (NIR) datasets was selected. NIR datasets were chosen because they are more common, offering a wider selection of images to test with.

The first dataset to be added to the generic platform was CASIA V1. This was the dataset used by Masek when he developed his iris recognition algorithm. With the LEI dataset not being publicly available, the CASIA V1 dataset was the only way to attempt to replicate Masek’s work. This use by Masek meant that in comparing Masek’s work with others, the CASIA V1 dataset should provide the baseline comparison.

The CASIA V3i contains all the images from CASIA V1 plus quite a few more, but none of them have been pre-processed [137] with modified pupil regions. Thus, the CASIA V3i provides the best real world test for any recognition algorithm developed using the CASIA V1 dataset. However, the CASIA V1 and V3i datasets are among the easiest iris images to process with clear, well focussed images from co-operative subjects looking directly at the camera.

The CASIA V2 datasets are much more challenging with more of the face included in the images and larger variations in the ambient lighting during image capture making these an obvious choice as a more difficult dataset. There are a further five image datasets from CASIA which were not used, the main reason for this is that Chinese eyes were already well represented and these other datasets, except CASIA-Iris-Distance (V3d) brought nothing new to the challenge of recognition that was already present in the V1, V3i, V2D1 and V2D2 datasets.

The CASIA V3d contains partial face images with both eyes, the nose, hair, ears and the mouth visible in the image. This dataset was disregarded because it added facial recognition, something that none of the algorithms under test were designed to cope with.

The CASIA V3L, V4t and V4s datasets were simply too big at 16,625, 19,980 and 9,990 images respectively. The time to test these datasets would be measured in weeks, not days, and would have had a significant impact on the ability to complete this research.

To put this in perspective, the UBIRIS V2 dataset was tested with segmentation three using match three, the fastest algorithm combination available to this research. These 11,101 images at 400×300 pixels took one day to encode and 18 days to analyse. During this research many algorithms were tested and each of the algorithms developed was tested multiple times taking between one and seven days each time. Adding even one of these large datasets would have increased that to 25 days per test.

The CASIA V3t dataset contains images that are similar in quality to V2D1 and V2D2 so it was unnecessary to include it as its features were already being tested by the V2 datasets. The CASIA image datasets are also predominately Chinese and Asian eyes, which tend to have a different shape and eyelash density than African or Western eyes so more variety was needed.

The IIT Delhi dataset was added as it is the only dataset with predominantly Indian eyes which are more rounded with darker irises than Chinese eyes. The two MMU datasets were also included as these contain significantly more challenging images across a diverse range of ethnic groups. The final dataset to be included in the test library was the WVU Free dataset. This dataset was used by Li for testing his iris recognition algorithm and also

contains challenging images across a selection of ethnic groups. The focus for the WVU dataset is on off angle irises, where the eye is not looking directly at the camera.

WVU Free was selected instead of the full WVU dataset because that is the exact set of images that Li used for his research [12]. While this is not the full dataset, using it provided Li's flow the best opportunity to perform well and gave a baseline from which to judge its abilities.

The UPOL dataset was not used because the images were too artificial, with no eyelashes or eyelids. The only challenge came from the fact that the apparatus introduced noise, in the form of the black border and strong central specular reflections in the pupil. This must be specially detected, but it is then straightforward to infill it with a neutral colour. The UPOL dataset is also a visible light dataset, not near-infrared.

The UBIRIS V2 dataset is too large, and both UBIRIS datasets are visible light datasets, so were not used. The ICE dataset is another challenging multi-ethnic dataset of NIR images, but it was discovered later in the research and added nothing new. At that time 9,560 images were being used for testing, so the additional images from the ICE dataset were not required.

The eight datasets used for this research were thus:

- CASIA V1,
- CASIA V2D1
- CASIA V2D2,
- CASIA V3i
- MMU V1
- MMU V2
- IIT Delhi
- WVU Free

As there are mostly Chinese eyes in the CASIA datasets, mostly Indian eyes in the IITD dataset and cross sections including European and American eyes in the MMU and WVU Free datasets, it was felt that this was a sufficient cross section of eye variations to allow valid comparisons to be made.

This selection of images provides a significant number of comparisons for the matching process with high numbers of inter-class matches (matching images from different eyes) and a large selection of intra-class matches (matching different images of the same eye). These are the two key comparison types for any biometric system.

With the benefit of hindsight, the WVU dataset was very challenging and therefore this research would have benefited from testing with the more statistically significant 865 images supplied with the full WVU dataset than the 80 supplied in the WVU Free dataset.

In addition, despite the inclusion of four datasets with mixed or Indian ethnicity, there is a significant bias toward Chinese eyes as 60.6% of the images come from this ethnic group via the CASIA datasets. For this reason, there would have been more variety if the CASIA V2D1 dataset had been replaced by the ICE dataset, particularly as the results collected during this research showed the CASIA V2D1 dataset to be one of the less challenging datasets.

However, the eight datasets used by this research is the widest variety and selection of datasets used in any of the literature found.

The next section presents the finished generic platform and discusses its use and various modes of operation.

3.8 The Generic GUI

Having defined the requirements for the GUI in the previous sections, this section looks at the finished GUI. A brief ‘Quick-Start’ user manual describing how to install the generic platform and get started using it is presented in appendix Appendix A.

Two types of processing are carried out as part of the normal development cycle of an iris recognition algorithm. The first type is a manual, iterative process of testing the algorithm against a small number of images, typically less than ten. During this process, the algorithm is refined with manual evaluation of the results. This phase is especially effective for the earlier stages of pre-processing, segmentation and normalisation.

The second type of processing is a bulk evaluation of the algorithm under test. Normally automated, this encodes large numbers of images and then conducts a thorough one-against-many evaluation of the iris code creation and recognition performance of the algorithm selected. The user must be able to select the desired process and then leave the generic platform to get on with the job of processing images without further user interaction.

For the first prototype, the manual and automatic flows as well as the analysis module were all separate GUI’s. However, the need to set the algorithm choices and options on two GUI’s was time consuming and error prone for the user. The need to maintain duplicate code and GUI’s made the work of developing the platform slower and more error prone than was necessary. Therefore, the generic platform was redesigned to be a single GUI.

The iris recognition process was previously defined in Chapter 2 as having these six steps:

1. Image Capture
2. Image Pre-Processing
3. Image Segmentation
4. Iris Normalisation
5. Iris Encoding
6. Iris code matching

Of these six steps in the iris recognition process only the first, ‘Image Capture’ is not implemented within the generic platform. However, the second step, ‘Image Pre-

Processing', is merged with the segmentation stage and is not explicitly mentioned or separated within the platform. The platform was designed like this because segmentation algorithms can often carry out pre-processing at the beginning of any one of the four boundaries of interest (pupil-iris, iris-sclera, top and bottom eyelids). Pre-processing can also be used before the normalisation step in order to remove specular reflections and eyelashes.

While this choice to leave the pre-processing merged into the segmentation simplified the user interface by having fewer GUI elements, with the benefit of hindsight it is clear that greater flexibility would have resulted from having the pre-processing as separately selectable options for each of the four segmentation boundaries as well as for the normalisation step.

The remainder of this section presents the generic platform GUI. Figure 3.12 shows the generic platform GUI in manual mode and Figure 3.13 shows the GUI in Automatic mode.

Each area of interest on the images is labelled with a large blue number. In the following text, these area numbers are referenced using curved brackets '()'. In manual mode (Figure 3.12) areas (6) and (7) are available. In automatic mode (Figure 3.13) areas (6) and (7) are replaced with areas (11) and (12).

For both Figure 3.12 and Figure 3.13, the pane shown in (8) is the segmentation pane which corresponds to the segmentation step. The normalisation pane, which is visible in (8) when the normalisation 'tab' is selected in (10) and corresponds to the normalisation step, is shown in Figure 3.16 on page 131. The encoding 'tab' is shown in Figure 3.18 on page 133 and the matching tab in Figure 3.19 on page 134.

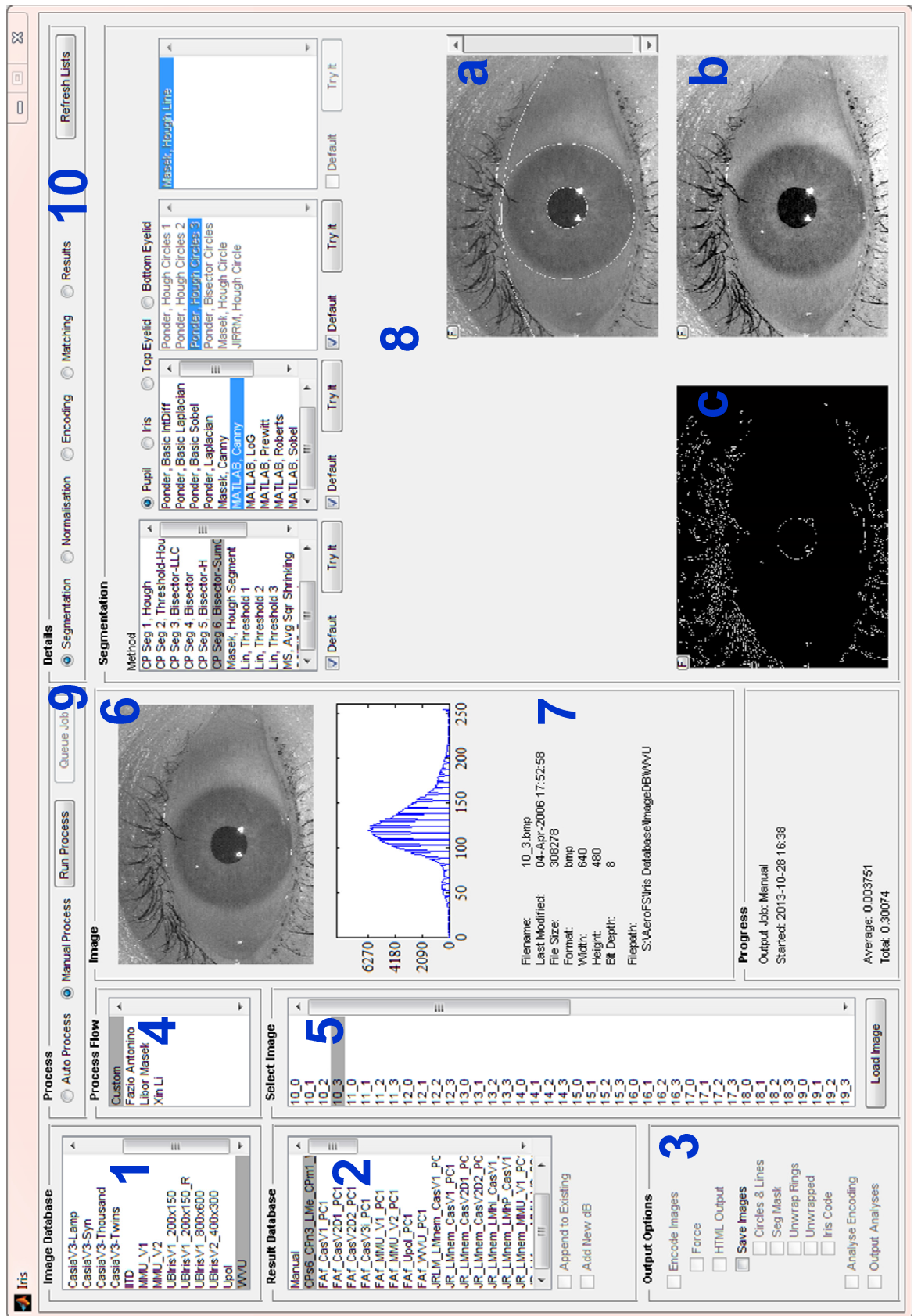


Figure 3.12 – Generic Platform GUI running PC1 with ‘Manual Process’ selected and GUI regions numbered in blue

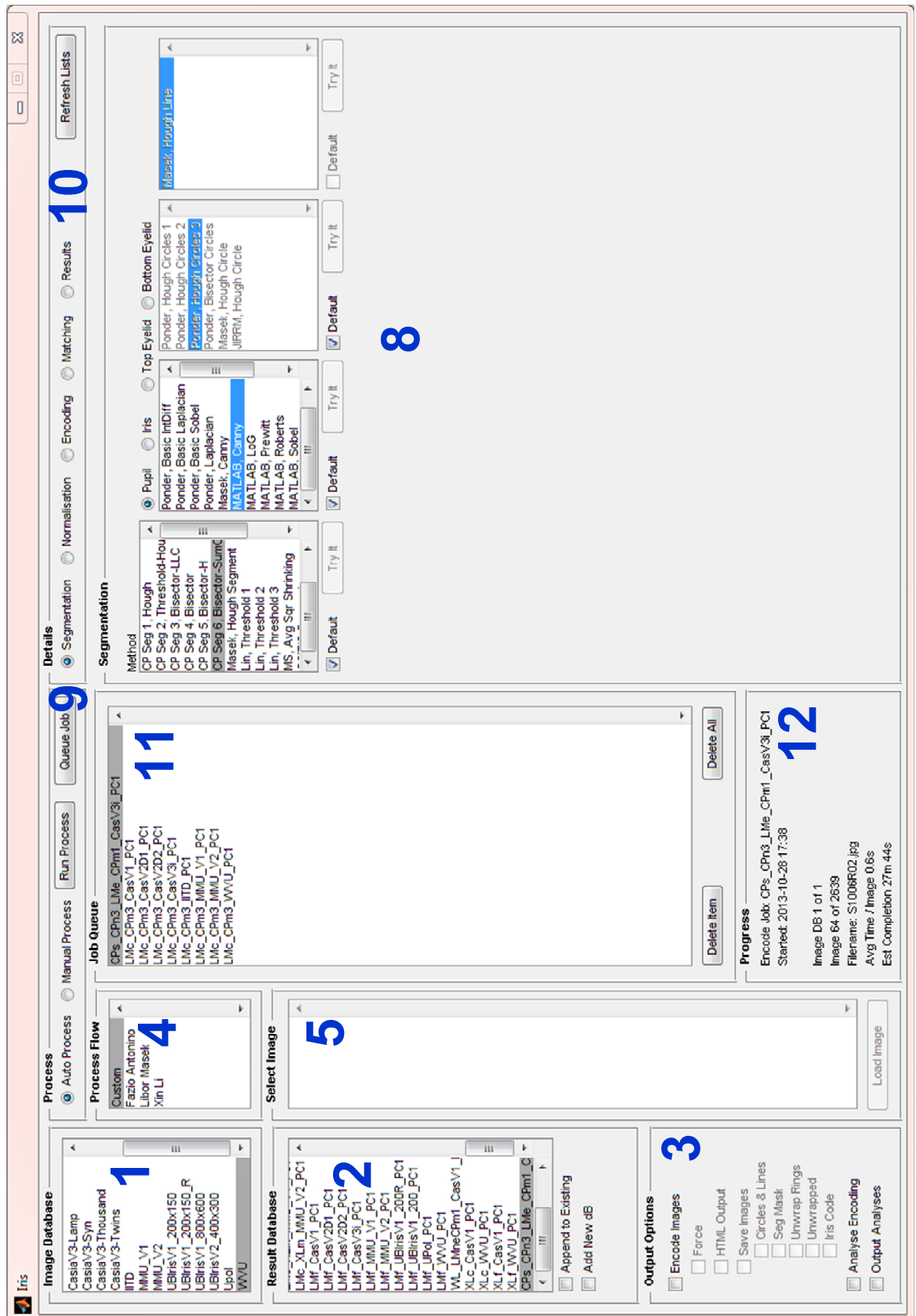


Figure 3.13 – Generic Platform GUI running PC1 with ‘Auto Process’ selected and GUI regions numbered in blue

Switching between automatic and manual modes is achieved by clicking the option buttons in area (9). In both modes the ‘Run Process’ button in area (9) will start the recognition process. In manual mode the results are displayed in area (8) under the relevant tabs for segmentation, normalisation, encoding and matching. Figure 3.12 shows the results of a segmentation process in area (8) with image (a) showing the image with segmentation boundaries, (b) showing the image pre-processed and image (c) showing the Canny edge detected image.

Two formats can be used when integrating an algorithm into the generic platform. The first format is the ‘fixed flow’, area (4) on Figure 3.12. When an iris recognition algorithm is adapted to execute within the generic platform as a ‘fixed flow’ it is being used with the minimal alterations being made to its code. The term ‘fixed flow’ refers to the lack of choice during the encoding stage. The ‘fixed flow’ contains the recognition process from pre-processing to encoding with no options to modify this in any way.

The input to a ‘fixed flow’ is a variable containing the iris image. The output from the ‘fixed flow’ is a variable structure containing the iris code and mask. The configuration options for fixed flows are the image dataset, the ‘fixed flow’ to use, the matching algorithm used for analysis, the name of the results output and the processing options such as whether to output segmentation images for reviewing. This selection process is shown in Figure 3.14. (blue text indicates figure number and area of figure for reference)

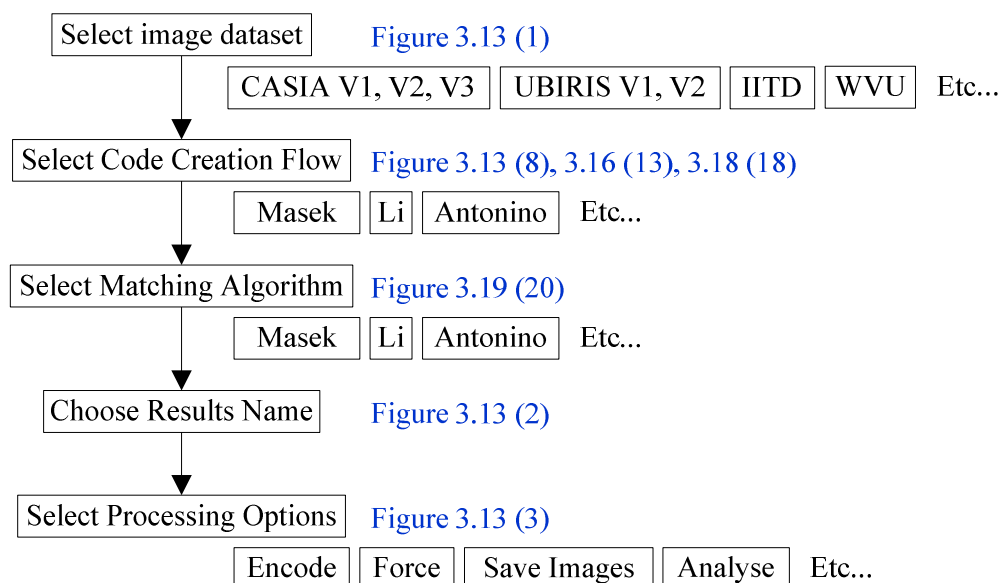


Figure 3.14 – Flow chart depicting the selection choices available options for iris recognition algorithms added as a fixed flow.

In the selection process for the ‘fixed flow’ in Figure 3.14, each of the connected boxes on the left of the diagram shows one of the five choices to be made with a selection of the choices available shown in the boxes grouped by each choice.

The second algorithm integration format is the ‘custom flow’. An iris recognition algorithm that is adapted to execute within the generic platform as a ‘custom flow’ has each stage of the recognition that it provides separated into the recognition stages with each stage included as a separate task accessible within the platform. Integrating algorithms fully into the platform is called ‘custom flow’ because it requires much more customisation of the original code in order to function within the generic platform.

Edge, line and circle detection and segmentation mask can also all be separated into separate modules. Each of these modules is listed within the GUI, Figure 3.12 (8) and Figure 3.13 (8), and is selectable by the user. These flows are both shown in the generic platform user experience flow chart, Figure 3.15 on page 130.

Using the ‘custom flow’ the user can, therefore, select which algorithm is to be used by the generic platform for each stage of the recognition process, allowing direct comparisons to be made between code by different authors. E.g. the same normalisation, encoding and matching algorithms could be used to test a variety of segmentation algorithms giving a direct comparison of the effectiveness of the segmentation algorithms. Thus, in the case where an author has only developed code for one of the four tasks, that single task can be tested and evaluated within a full design flow.

The image dataset (1) and results database (2) are selected by the user, along with the flow method (4). If ‘Custom’ is the selected flow, then (8) is enabled for algorithm selection for the Segmentation, normalisation and encoding stages.

Area (10) is used to change area (8) between these functions in the manner of tabs used in modern GUI’s. This form of implementation is used as there are no native tabs in MATLAB’s GUIDE system. If ‘Custom’ is not selected then only the matching and results selections in area (8) can be accessed. This section also contains the “Refresh Lists” button that reloads the image datasets and modules when pressed, removing the need to restart the platform when a new module or image dataset is added.

Area (8) allows algorithms from different researchers and developers to be selected, providing a simple click selection from lists to change the algorithms used. This is one of the core features of the generic platform, making it very flexible and simple to use.

This flexibility goes further than just switching core algorithms for segmentation, normalisation, encoding and matching. If the algorithm is coded to support it, the user can switch in or out sub functions, for example during the segmentation operation, different edge, circle or line detection can be used to see the effect of these on the recognition rates. The 'Try it' buttons under each list box allow the selected algorithm to be tested quickly.

Images displayed in area (8) of the GUI can be opened in a separate figure window by clicking on the small button labelled 'F' in their top left corner. This allows the images to be resized and magnified as required.

In manual mode area (5) lists all the images in the currently selected dataset, area (6) displays the currently selected image along with its histogram and area (7) displays detailed file information.

Figure 3.13 shows the GUI in automatic mode with the Queue Job button enabled. The Queue Job button allows the current configuration to be added to the job list (11). This allows the developer to set up a variety of configurations of iris recognition process to be queued up. As long as each configuration is given a different result database name (2) when the Run Process button is pressed the platform will run each method one after the other.

Area (12) on Figure 3.13 is used to display the progress of the current job. This contains the name of the job, the progress through the job and an estimate of how long the job will take to complete. In the event that a job is unable to complete, the error output to the MATLAB command line will include the contents of the progress panel.

In automatic mode the desired output options (3) can be set. These options are partially intelligent, in that if an encode database already exists then the user can skip that step and only do the analysis or even just the output step. The save images check box enables the image type check boxes allowing the user to select which debug images they would like to

save to disc. The 'Force' check box allows the user to stop the platform using any pre-calculated data and forces it to calculate everything.

The user experience flow chart for the GUI is shown in Figure 3.15. It clearly shows that there is a core synergy between the automated and manual process flows. It also shows the 'fixed flow' as compared to the custom flow. (blue text indicates figure number and area of figure for reference)

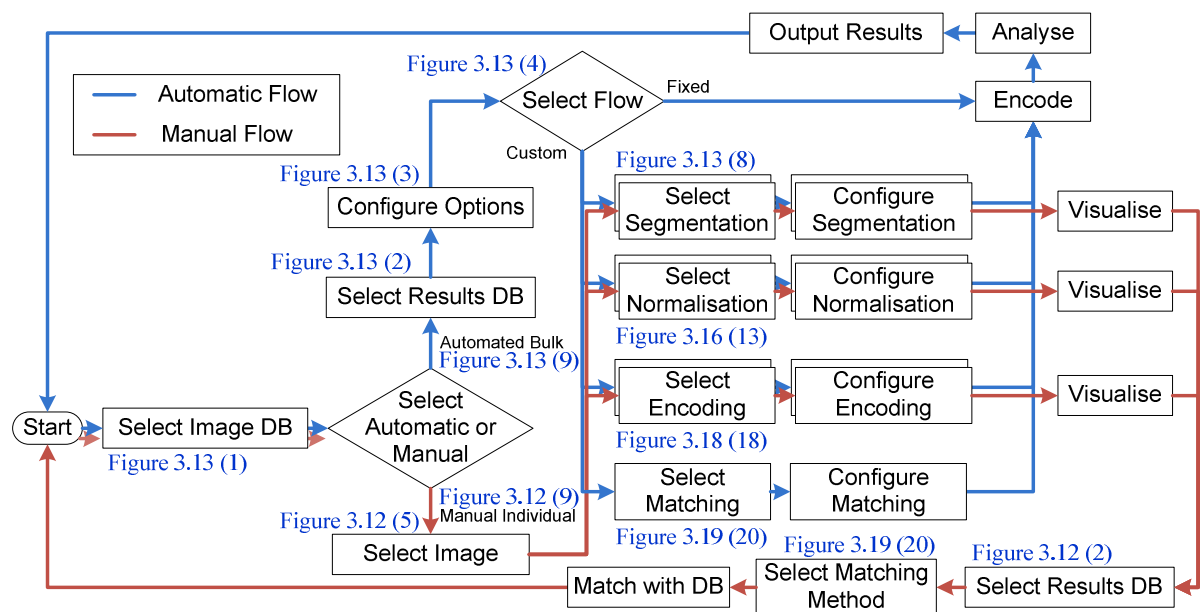


Figure 3.15 – Generic Platform User Experience Flowchart

The first step for the user is to select an image database to use (1). The GUI will default to automatic mode, but the user can switch to manual mode at any time (9). In automated mode (Figure 3.13) the user will select or add a results database (2), configure the required options (3) and then select the flow (4). If any of the named flows, i.e. not 'Custom', are selected then the configuration is complete and the user can either select 'Run Process' to run the process that is configured, or 'Queue Job' to add the process to the job queue for processing later.

If the 'Custom' flow is selected (4), then the user is able to select the segmentation, normalisation, encoding and matching algorithms using areas (8) and (10). Configuration is then complete and the user can either select 'Run Process' to run the process that is configured, or 'Queue Job' to add the process to the job queue for processing later.

For both custom and fixed flows in automated mode the platform will then encode all the images in the image dataset and then performs a one against all comparison of each iris code to find the best match. Finally the results will be output to the results database.

In manual mode (Figure 3.12) the required image must be selected from (5) and it will be displayed in (6) with detailed information about the image displayed in (7). The user is then able to select the segmentation, normalisation, encoding and matching algorithms using areas (8) and (10).

Figure 3.12 shows the segmentation pane selected with an iris image segmented. In manual mode the ‘Try it’ buttons are enabled allowing just the segmentation stage, or even just the edge detection to be run. The remainder of this section discusses the other tabs in area (8)

Figure 3.16 shows the normalisation pane from the generic platform in manual mode.

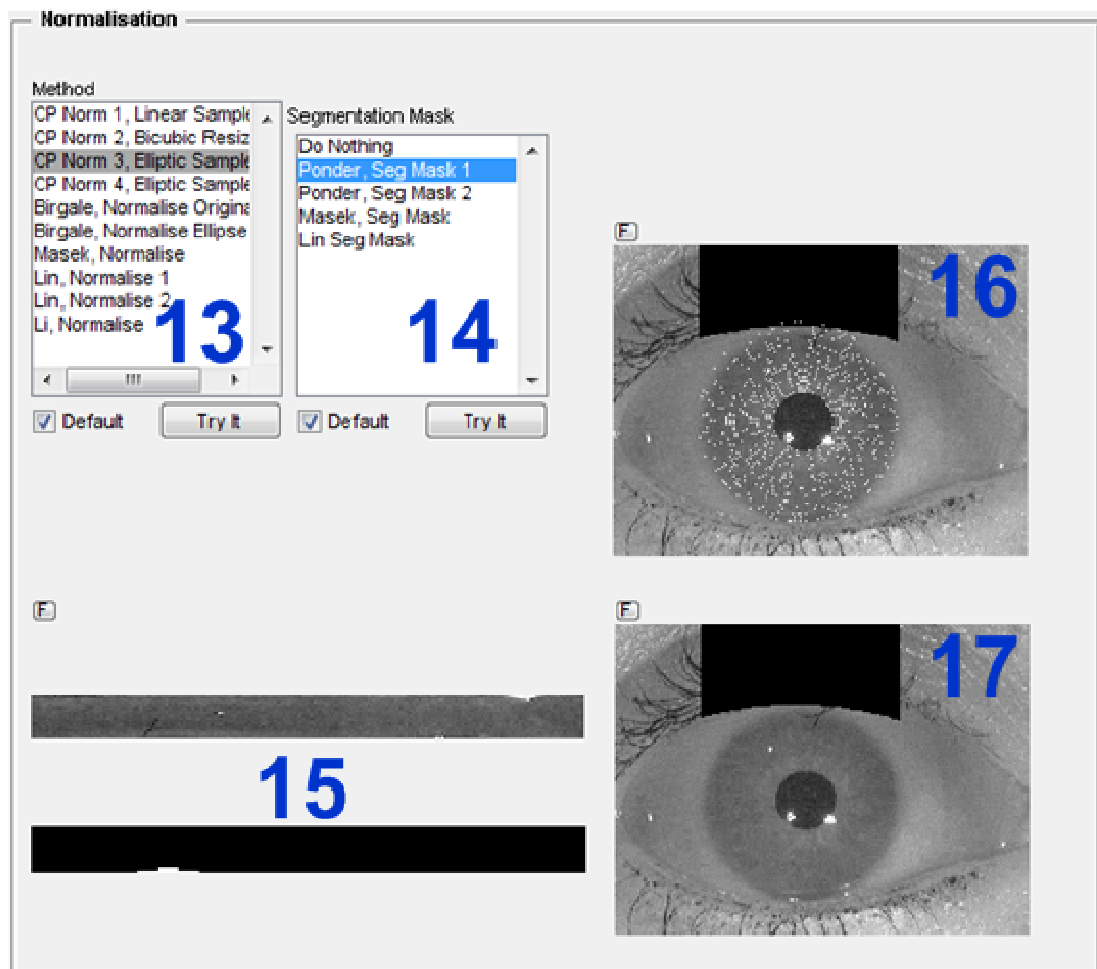


Figure 3.16 – Generic platform normalisation pane showing a normalised iris with its mask and two debug images.

In Figure 3.16 the method list box (13), is the list of available normalisation routines. The segmentation mask list box (14), contains the available segmentation mask routines. Clicking on a normalise routine in the method list will automatically select its default segmentation mask or the segmentation mask routine that has been previously selected by the user.

The two images at (15) are the unwrapped iris (top) and its mask (bottom). The black areas of the mask show the parts of the unwrapped iris that will be included in the encoding and matching processes.

The image at (17) shows the original image with the segmentation mask applied. In this instance the top eyelid does not occlude the iris by very much, as also seen in the white area of the mask in (15). The black masking is only applied to the eye for the width of the outer iris boundary in order to maximise performance. The image at (16) shows the segmentation mask with the iris sample points overlaid.

For display in the GUI the images are automatically resized by MATLAB from their original size down to 260x195 pixels using the bicubic interpolation method. For bicubic interpolation each pixel in the destination image is a weighted average of pixels in the nearest four-by-four neighbourhood on the source image.

Unfortunately, image re-sizing tends to lose much of the desired detail, especially as the images get larger. Even for the CASIA V1 images at 320x280 pixel corruption occurs, for the higher resolution images, like those shown in Figure 3.16 area (16) whose original resolution was at 640x480, the problem is worse.

Image (16) clearly shows this problem as the visible white spots should show as concentric rings. For this reason, each image in the platform has a small 'F' button which opens a separate figure window when pressed. The figure window will display the image at its original resolution. Figure 3.17 shows the image at (16) when opened in a separate figure window; the concentric rings can clearly be seen.

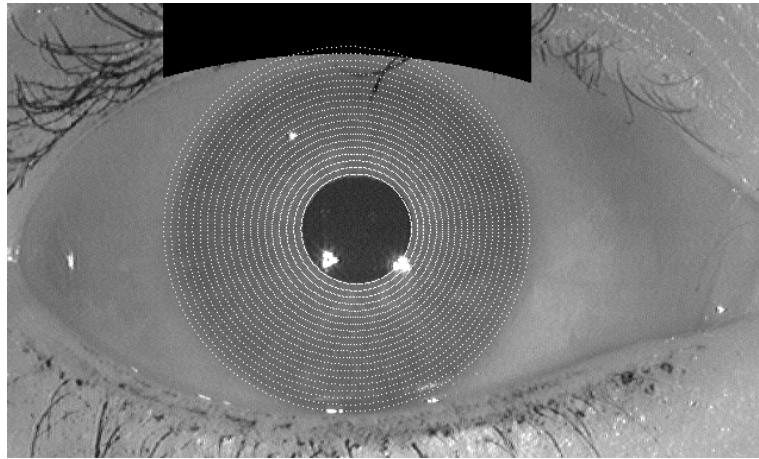


Figure 3.17 – Image (16) from Figure 3.16 when opened in a separate figure window

Figure 3.18 shows the encoding pane from the generic platform in manual mode with an encoded iris and its associated mask.

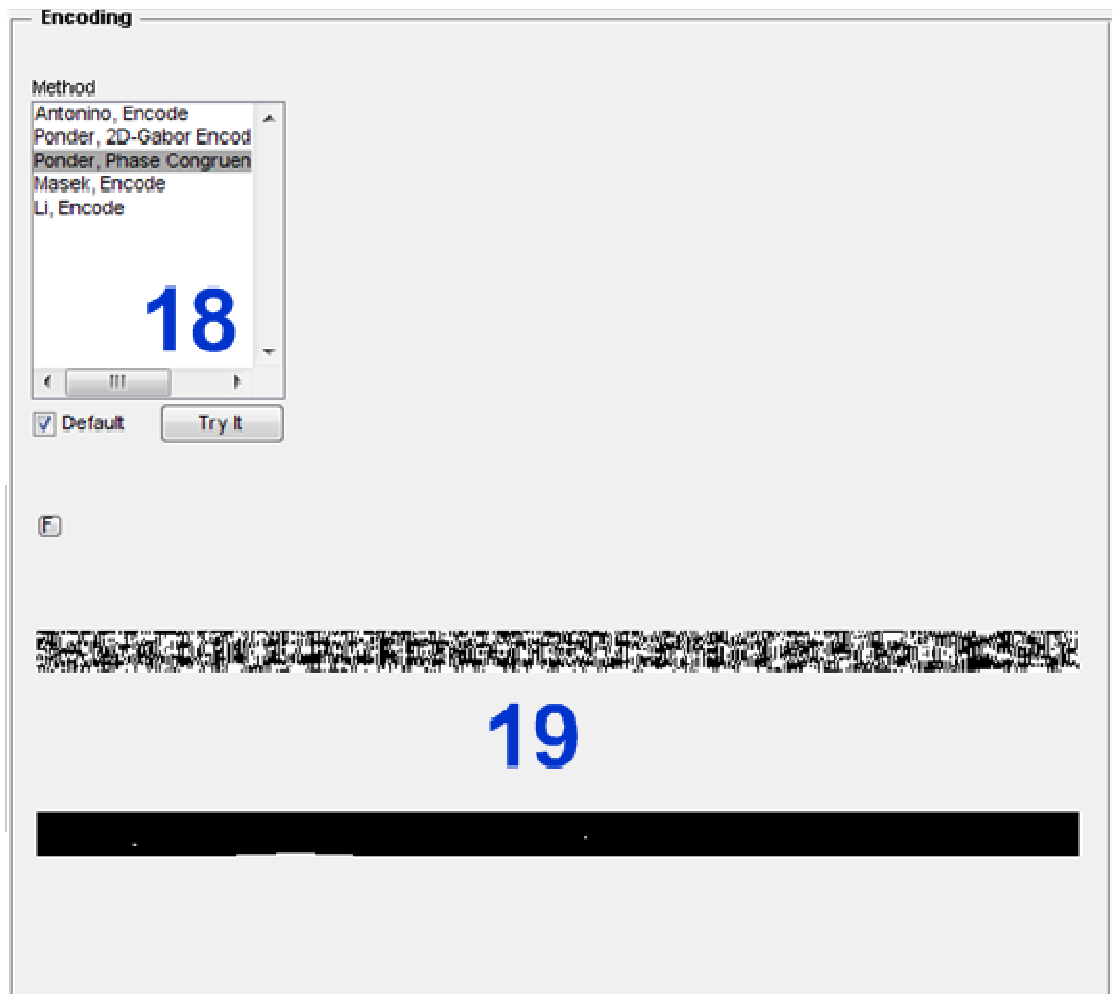


Figure 3.18 – Generic platform encoding pane showing an iris code with its mask

The list box at (18) in Figure 3.18 contains the list of available encoding algorithms. The images at (19) are the iris code (top) and its mask (bottom). Due to the need to keep the panes the same size, so they behave in the manner of tabs, this pane is mostly empty space as is the one on matching that follows (Figure 3.19).

Figure 3.19 shows the matching pane from the generic platform in manual mode with the results from matching the currently selected image, in this case 10_3 from the WVU Free dataset.

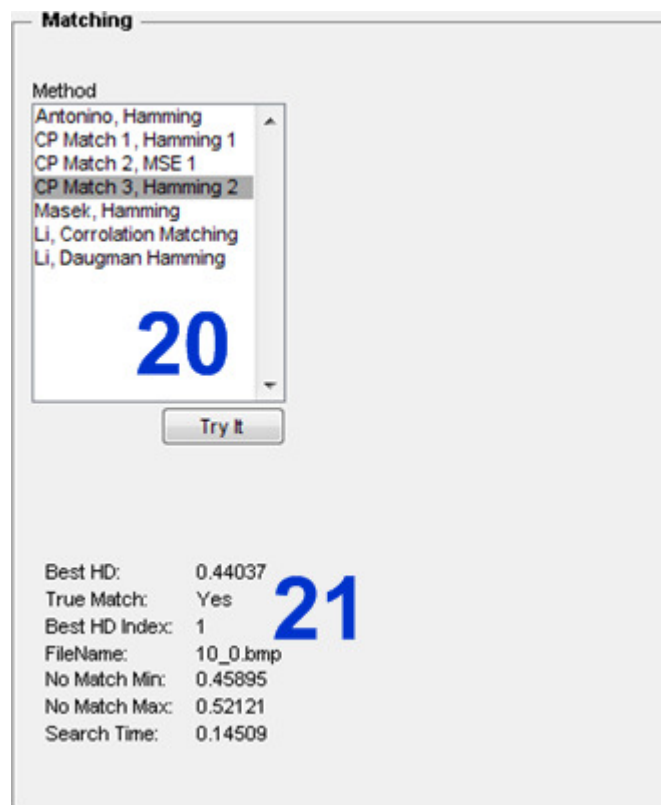


Figure 3.19 – Generic platform matching pane showing the results from matching the currently selected image (10_3.bmp from WVU Free dataset)

In manual mode, the matching pane in Figure 3.19 allows the user to compare the currently selected image with the currently selected results dataset (see Figure 3.12 area (2)). The results are displayed in area (21). In both automatic and manual modes, the method list box at (20) enables the desired matching algorithm to be selected.

Figure 3.20 shows the results pane from the generic platform in automatic mode with the reference segmentation boundaries applied to image 010204 from the MMU V2 image dataset, area (22). Area (23) lists the parameters of the various circles, as well as the best lines for this image. The check box in area (26) allows the user to swap to the reference segmentation (see Section 5.4) and the option buttons allow a grade to be selected for each boundary.

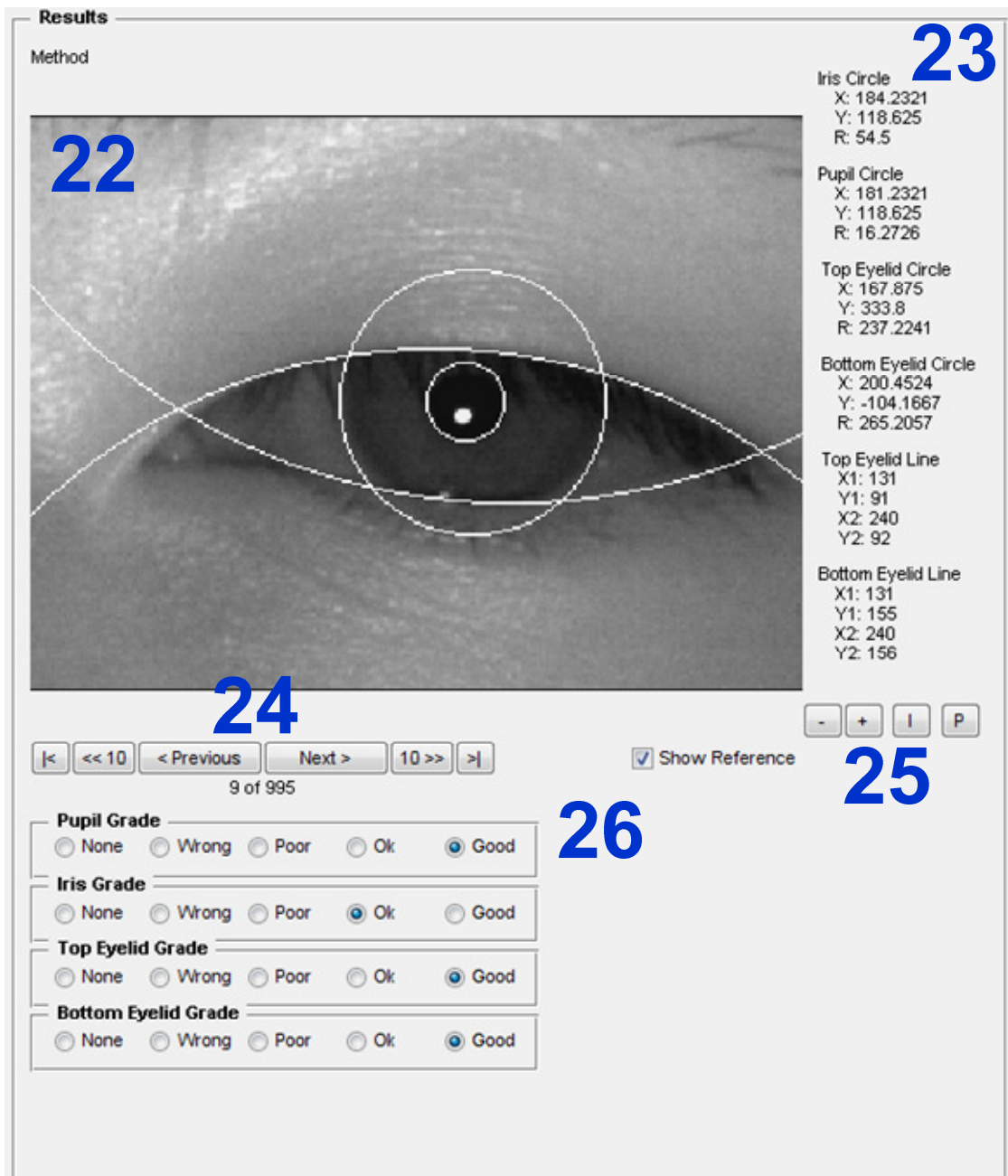


Figure 3.20 – Generic platform results pane showing the reference segmentation for image 010204 from the MMU V2 image dataset.

Area (24) allows the user to move the results displayed, and area (25) allows the user to zoom in and out with the I and P buttons zooming straight to the boundaries for the iris (see Figure 3.21) and pupil (see Figure 3.22).

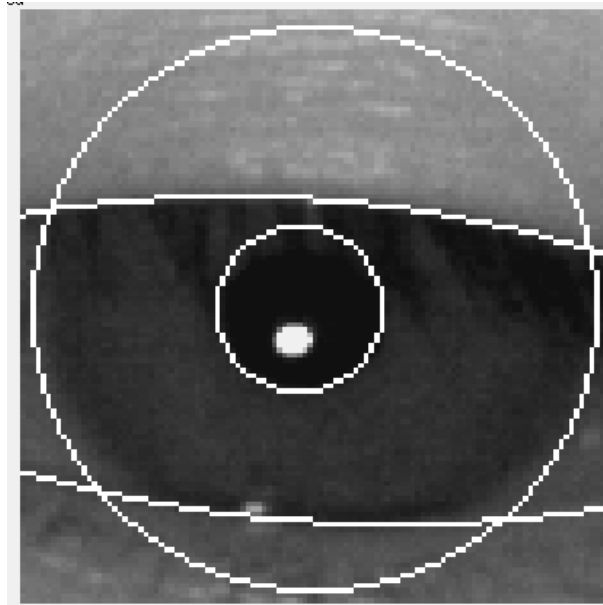


Figure 3.21 – Area (24) in Figure 3.20 after selecting the ‘I’ button to zoom in to the iris boundary

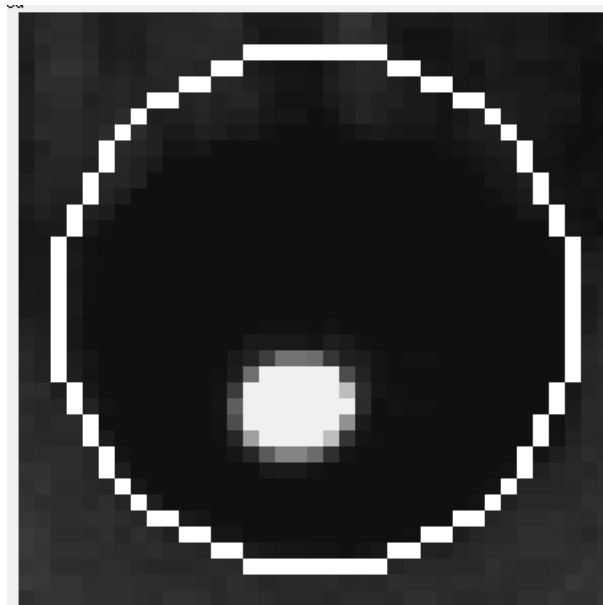


Figure 3.22 – Area (24) in Figure 3.20 after selecting the ‘P’ button to zoom in to the pupil boundary

The next section provides overall conclusions for this chapter.

3.9 Conclusions

At the start of this chapter, it was stated that the platform had to meet these criteria:

- It had to include a development environment that promoted fast development and be usable on different operating systems. (Section 3.1)
- It had to be able to output results in a form that would provide consistent, measurable, feedback of algorithm performance, thus, facilitating improved understanding of iris recognition (Section 3.2)
- It had to be possible to load and / or add different iris recognition operations without re-writing or recompiling the platform. (Section 3.3)
- It had to have all the different operations required for iris recognition in-built, but the default operations had to be replaceable with user-defined operations (Section 3.4)
- It had to allow the user defined operations and the in-built operations of the same type to be interchanged quickly and easily to facilitate process evaluation (Section 3.5)
- It had to provide the ability to quickly and easily load multiple image datasets into the GUI (Sections 3.6 and 3.7)

During this chapter, the available choices and reasoning behind the choices selected was discussed.

Selecting MATLAB as the language and development environment provided the fast development cycle and cross-platform capability required. This choice has its problems, not least of which is the high cost of ownership of MATLAB. However, the requirement was to create a very complex system, then to use that system to evaluate the performance of different iris recognition algorithms and develop improved algorithms. The timescale for all this work was fairly short, so making use of professionally produced and maintained software like MATLAB may still have been the best overall decision.

The decision to maximise the result reporting facilitated a far greater understanding of how each algorithm performed when compared to other algorithms and provided the consistent measurable feedback required. This ensures that the generic platform is fully capable of providing the evaluation and benchmarking necessary for the task of developing an iris recognition algorithm.

By breaking the recognition process into the functional blocks of segmentation, normalisation, encoding, matching, finding a line, finding a circle, edge detection and segmentation mask creation the first of these was achieved. So long as others have provided functions within all the required functional groups, the user only need create a function to replace one that is already there.

The layout of the GUI allows fast swapping of these functions within the recognition process. The code must be written to support the relevant function being swapped, but once that is done, the GUI takes care of the rest. This allows a very flexible development and research-orientated environment.

The use of directories for image datasets, results databases and modules provides a very easy and fast method of managing data. By having the platform find the images, databases and functions within the modules, the user is freed from time-consuming data management tasks, so long as the basic requirements are met.

The available image datasets were described and the reasoning behind the selection of the final eight datasets used for testing was discussed.

The next chapter evaluates the performance of existing open-source algorithms as well as algorithms from the literature and supporting operations such as edge detection.

4 Evaluating the Generic Platform

In this chapter, the process of evaluating the generic platform [112] is discussed. In Sections 4.1 and 4.2 Masek's [29, 31] complete iris recognition flow from pre-processing to matching is first to be evaluated as it was the code around which the platform was developed.

Masek's work was used because it was the most widely referenced in the literature, [32, 33, 34, 35, 36, 37, 87, 88, 89, 86]. It was also the first open-source iris recognition algorithm. The clean, well-structured code, which is divided neatly into the main four stages of recognition, was an ideal base to build upon.

The four remaining open-source iris recognition algorithms available at the time the research was carried out are integrated into the generic platform and evaluated. Methods introduced by Li [12, 27], Antonino [38], Sutra *et al.* [39] and JIRRM [40] are either completely or partially converted into modules for execution and evaluation within the generic platform in Section 4.3.

These four methods were investigated to see the state of the art within the open-source iris recognition field. Antonio's algorithm was of interest because it was a derivation of Masek's work replacing just the encoding and matching algorithms. Li and Sutra's algorithms are, like Masek's work, 'fixed flow's containing all the recognition stages. JIRRM was of interest because it also uses the Hough transform [41], which, in Java at least, executed very quickly and it was interesting to determine whether the algorithm would perform faster or slower than Masek's Hough transform code within the MATLAB environment.

The remaining three algorithms from Rosa [42, 43] and GruSoft [44] are then investigated and the reasons for not converting them to run under the generic platform are discussed.

Table 4.1 summarises all the iris recognition algorithms available for download at the time this research was carried out, in the order that they are addressed in this chapter.

Table 4.1 – Iris recognition algorithms available for download

Name	OS	Programming Language	Source
Masek's System	Windows / Mac OS / Linux	MATLAB	Yes
IRIS-ICT	Windows / Mac OS / Linux	MATLAB	Yes
Iris by Xin Li	Windows / Mac OS / Linux	MATLAB	Yes
JIRRM	Windows / Mac OS / Linux	Java	Yes
OSIRIS	Linux	C/C++	Yes
Iris Recognition System	Windows / Mac OS / Linux	MATLAB	Paid for
High Performance Iris Recognition	Windows / Mac OS / Linux	MATLAB	Paid for
GIRIST	Windows	C/C++	No

Three segmentation methods from the literature are implemented and evaluated. These include the segmentation methods introduced by Lin *et al.* [30], Lu [60] and Shamsi *et al.* [62]. The method from Lin *et al.* was selected because at that time it was a recent paper that used the same iris image dataset as Masek and Lu. The method from Shamsi *et al.* was selected as the only method from the literature that used the circle bisector method.

Section 4.6 adds supporting functions for edge and circle detection and tests these alternative routines in place of the default ones in Masek's algorithms. This allows the recognition and processing performance differences obtained from varying these functions.

There are four performance concerns when using the MATLAB environment, and those are considered in the final section of this chapter. First, the performance of algorithms used as a 'Fixed Flow' is compared to that obtained when running the algorithms as a 'Custom Flow'. How execution times within the generic platform might translate to real world performance is then discussed.

The third concern investigated is the performance effects of using different operating systems and computer hardware. During the course of this research, a number of

computers were used to develop and evaluate the generic platform and iris recognition algorithms; these are listed in Table 4.2.

Table 4.2 – Test machines used to develop and evaluate the generic platform and iris recognition algorithms

Test Machines				
Name	PC1	PC2	PC3	PC4
OS	64-bit Win7	64-bit Win7	64-bit LinuxMint 10 (Gnome Ed.)	32-bit WinXP
CPU	Core i7-2600K @ 3.40GHz	Core2Duo T9900 @ 3.06GHz		Pentium 4 @ 2.60GHz
GPU	nVidia GeForce GTX 550 Ti	nVidia Quadro NVS 160M		ATI Radeon 9250
RAM	8GB	4GB		1GB
MATLAB Version	r2009a			r2006a & r2009a

PC2 and PC3 from Table 4.2 were used for all the initial development activities for the generic platform. These were the same physical hardware but different operating systems, Windows 7 and Linux Mint 10. PC1 was assembled later on and was used thereafter for all development and research.

Windows 7 and LinuxMint 10 were the latest versions of these operating systems at the time the research outlined in this thesis was started. Newer versions did become available during the research, but for consistency the original OS's were kept. PC1 was very close to the state-of-the-art when it was assembled. The Core i7 processor was two speed grades down from the fastest processor available and the GeForce GTX 550 Ti was similarly two speed grades from the fastest graphics card from nVidia at that time. The hardware in PC2/3 was three years old at this time and the hardware in PC4 was ten years old.

All execution times presented in this chapter are from PC1 unless stated otherwise, as this provides the most consistent presentation of the information and allows results to be compared easily. Section 4.6 describes the performance differences between the various machines.

Finally, a method of accelerating the task of evaluating iris recognition algorithms within the generic platform is described and evaluated. The algorithms and methods discussed in this chapter are each dependent upon the previous. For this reason, each method will be described individually, with the results obtained by it, and then the next method will be described.

The separability results for the IITD dataset are consistently 0.0% for every single algorithm tested without exception. This appears to be due to an error within the dataset having two copies of the same image labelled as being different eyes. The problem that this presents is that the Hamming distance for the two images will be zero, meaning identical, but the true match routine will state that they must be different eyes and so will log a false match.

Even though a single erroneous result can invalidate the separability results, it will have minimal measurable impact on the true match, decidability and EER results, so these are still presented. There was insufficient time to track down the erroneous image so the separability result for the IITD dataset is ignored in the text and any separability averages presented.

The code creation and analysis times presented in this chapter are, unless stated otherwise, measured by MATLAB but are dependent upon both the algorithm implementation as well as the system load. As MATLAB code executes much slower than real-time speeds real world evaluation of the performance is difficult, however, the timings are suitable for the comparison of the performance of one algorithm run within MATLAB against another algorithm also run under the same MATLAB environment.

The next section briefly looks at the initial development of the platform, then at the initial testing of Masek's iris recognition algorithm within the generic platform and comparison of the results obtained using the generic platform with those in the literature.

4.1 Developing the Generic Platform with Masek's Iris Recognition Algorithm

This section looks at the development of the platform and the initial testing using Masek's code [29] which was used as the core around which the first generation, three-part GUI was developed. When the testing of Masek's code was complete, the three GUI's were merged into one GUI, considerably simplifying the development and evaluation process. Masek's work is the most widely referenced open-source iris recognition algorithm in the literature, [32, 33, 34, 35, 36, 37, 87, 88, 86, 89].

The key features of Masek's code are that it is very easy to use and the source code is available, allowing iris recognition researchers to investigate and experiment. It was also the first open-source iris recognition algorithm. Masek's code, with its picture-input, template-output approach, provides a good template for how to implement a very simple system.

Masek uses Canny edge detection [45] and Hough transforms [41] to segment the eye images. The Hough transform is described in more detail in Appendix Appendix C. A Daugman style polar unwrapping of the iris provides the separated iris image that is then encoded by convolving it using 1D Log-Gabor wavelets [31, 97]. Lastly, it is digitised by phase quadrature [76]. Matching is via the Daugman Hamming distance test [2].

Masek's code is separated cleanly into the four main iris recognition operations of segmentation, normalisation, encoding and matching. Pre-processing is carried out in the segmentation stage. The first three stages are linked using a top-level script called `createiristemplate`. However, no database or bulk comparison code is provided with the system.

As discussed in Section 3.8, The Generic GUI, pre-existing iris recognition algorithms can be integrated into the generic platform as either a 'fixed flow' or a 'custom flow'. The 'fixed flow' minimises changes to the code of the algorithm but in this form, the generic platform only provides matching and dataset customisation with standardised results output.

For the 'Fixed flow', the iris image to be encoded is passed to the recognition code, which is then expected to segment, normalise and encode the iris before returning an iris code and mask. The selected matching algorithm is then used for the analysis. This method has no ability to swap algorithms, but the process makes very few changes to the original code of the recognition algorithm.

The 'custom flow' enables much greater levels of flexibility for algorithm swapping but requires larger changes to be made to the algorithm's code for it to work. This separates the individual stages of segmentation, normalisation and encoding, and changes the input data and result data to be what the platform expects.

Customising an iris recognition algorithm to be used as a 'custom flow' within the generic platform's framework normally involves breaking it into four tasks: segmentation, normalisation, encoding and matching. Currently any image pre-processing that is needed must be carried out within the segmentation module. This a very invasive process requiring the correct inputs and outputs for each stage and sub-stage of the iris recognition algorithm. It is also the most flexible flow allowing full customisation of the recognition process.

The concern with these two methods of integrating iris algorithms into the generic platform was that the additional data passing and function calling inherent in the more flexible 'custom flow' would result in slower overall execution times than the 'fixed flow' implementations. Thus, Masek's code was implemented as both 'fixed' and 'custom' flows to allow a comparison of the execution times to be carried out.

Masek's code was already producing the required data for the 'fixed flow', but the data was returned as a mixture of saved files and returned variables. By changing the format of the data returned from `createiristemplate` from this mixture of saved files and returned variables to returning all the data in variables, it was possible to use this routine as the top level for the Masek 'fixed flow'. This data format adjustment was repeated for the matching routine `gethammingdistance`, enabling it to be used for the analysis stage of Masek's flow.

Breaking Masek's code up to allow it to be used as a 'custom flow' was a straightforward process requiring only adaptation of the input and output structures. This allowed the first

results to be returned by the platform using the CASIA V1 dataset for Masek's code running as a flow and as a custom flow.

Template files are provided for all the functions to help speed up the development of recognition tasks. These template files already include all the basic structure necessary to function within the generic platform, allowing the developer to concentrate on developing their algorithm.

4.1.1 Results

The results in this section compare the results presented in Masek's thesis with those obtained using the generic platform running Masek's iris recognition algorithm as both a 'fixed flow' and a 'custom flow'.

The results present seven metrics upon which the algorithm performance can be compared. The true match results show the percentage of images for which the best match found was an image of the same eye (Authentic Accept by Daugman's measure). The false match results show the percentage of images for which the best match found was an image of a different eye (Imposter Accept by Daugman's measure).

The separability figure shows the percentage of true matches for which the Hamming distance is lower than the lowest Hamming distance attained by the false matches. Separability is calculated using equation 4.1.

$$\text{Separability} = \frac{\sum(\text{True Matches} < \min(\text{false match}))}{\sum(\text{True Matches})} \times 100 \quad 4.1$$

For the perfect algorithm, there would be no overlap between the true matches and false matches, so the result of equation 4.1 would be 100% of true matches with a Hamming distance less than the lowest false match Hamming distance.

Decidability as a metric was proposed by Daugman [81], and is a number that represents how well grouped the inter-class and intra-class results from a recognition algorithm are. This gives very similar information to separability but reduces the impact of outliers.

Together separability and decidability give a much more complete picture of the quality of the results achieved than they do individually. If both separability and decidability are high then the results from an algorithm are well grouped within each class and setting a threshold will be straightforward, if both are low then the algorithm fails to separate authentic from imposters.

The Equal Error Rate (EER) is the percentage at which the False Accept Rate (FAR) and False Reject Rate (FRR) are the same value. The false reject rate (FRR) is the number of irises presented to the system that should be accepted, but are actually rejected. The false accept rate (FAR) is the number of irises presented to the system that should be rejected, but are accepted instead.

The FRR and FAR are often presented together on a single graph with percentage on the vertical (y) axis and Hamming threshold [100, 76] along the horizontal (x) axis. They are plotted as cumulative percentages at each threshold along the x axis.

The average code creation time is the average time it took Masek's segmentation, normalisation and encoding routines to convert an iris image into an iris code and mask set. The total code creation time is the total amount of time taken by Masek's segmentation, normalisation and encoding routines to convert all of the images in the dataset into iris codes with masks and store them in a database.

The total analysis time is the time taken for the analysis code in the generic platform to perform a one-against-many comparison of all the iris codes with masks using Masek's Hamming distance [100] based matching algorithm. Masek does not present true match, average code creation time, total code creation time or total analysis time in his thesis, so these have been left blank.

Table 4.3 compares the results from Masek's thesis against the results obtained by running the adapted code as a 'fixed flow' and as a 'custom flow' on the generic platform using the full CASIA V1 dataset. Ideally, all three columns should be the same, as the code and tests being run should in theory be the same.

Table 4.3 – CASIA V1 results, Masek's claimed results versus generic platform (GP) results in both 'fixed flow' and 'custom flow' form.

CASIA V1			
Method	Masek Thesis	Masek GP Flow	Masek GP Custom
True Match	-	98.0%	98.0%
Separability	99.8%	61.1%	61.1%
Decidability	6.2	3.6	3.6
Equal Error Rate (EER)	0.1%	4.1%	4.1%
Average Code Creation Time (s)	-	6.46	6.43
Total Code Creation Time (h:mm)	-	1:21	1:21
Total Analysis Time (h:mm)	-	1:11	1:14

The results shown in Table 4.3 show very clearly that the recognition results using Masek's code within the generic platform are unaffected by the choice of 'fixed flow' or 'custom flow' as the method of inclusion. The recognition results for 'Masek GP Flow' and 'Masek GP Custom' are the same, within measurable tolerances. Further testing was carried out to corroborate these results; this testing and the results achieved are discussed in Section 4.7.

The Separability result in Table 4.3 for Masek's algorithm running within the generic platform, is significantly lower at 61.1% when compared to the result of 99.8% presented in the literature. The same is true for the decidability at 3.6 instead of 6.2. The EER at 4.1% instead of 0.1% is a particularly significant increase. These results all show an algorithm that is performing worse within the generic platform than previously seen.

The generic platform encoding and analysis times in Table 4.3 show minor variations that were investigated further by repeating the tests 5 times. The times recorded varied on every run of the tests with average encoding times from 6.36 seconds to 6.50 seconds. Total database (DB) encoding time was consistently between one hour 20 minutes and one hour 22 minutes and the database analysis time was between one hour ten minutes and one hour 21 minutes.

There was no reason within the generic platform for this variation, but it was noted that the host system load varied on each test run. Thus, it was concluded that these minor variations in execution time were due to system load variation, rather than inconsistencies within the generic platform or any difference between the 'fixed flow' and 'custom flow' mechanisms.

The results in Table 4.3 also show a distinct reduction in the separability, decidability, equal error rate (EER) and EER Hamming threshold results achieved by Masek when compared to the results obtained using Masek's code within the generic platform.

Figure 4.1 shows the FAR against FRR graph for Masek's results using the CASIA-a dataset, as detailed in his thesis [31], side-by-side with the FAR against FRR graph for Masek's code when executed within the generic platform using the CASIA V1 dataset.

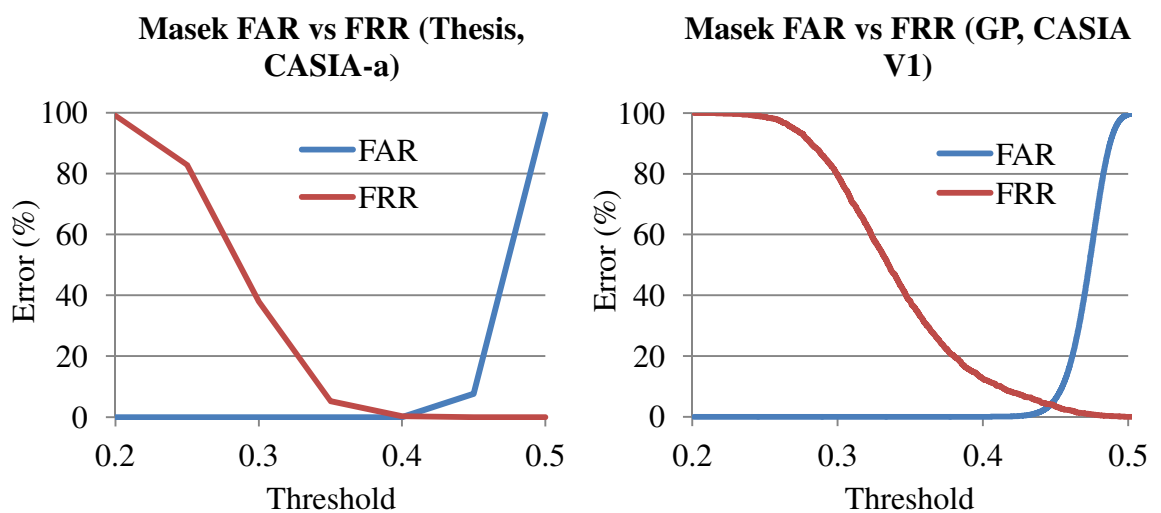


Figure 4.1 – FAR and FRR curves for Masek's thesis results using CASIA-a image dataset (Left) and Masek's algorithm (Right) when executed within the generic platform using CASIA V1 image dataset

Each figure comprises two graphs, one of FAR against threshold, and the other of FRR against threshold. These are shown on the same graph in order to find the point at which these cross, which is the optimum Hamming distance at which the number of falsely accepted irises is the same as the number of falsely rejected irises. This point is the equal error rate (EER).

The two graphs in Figure 4.1 are presented side-by-side for easier comparison and should be the same, but they are different. This difference is also seen in the Equal Error Rate (EER) and EER Hamming Threshold in Table 4.3. The reasons for these differences became apparent upon re-reading Section 5.2.3 of Masek's thesis:

“5.2.3 Actual Data Sets

It was not possible to use all of the eye images from each database, since perfect segmentation success rates were not attained. Instead, a sub-set of each database was selected, which contained only those images that were segmented successfully. The details of each sub-set are outlined in Table 5.1. [Table 4.4]”

Table 4.4 – Actual image datasets used by Masek

Set Name	Super Set	Number of Eye Images	Possible Intra-Class Comparisons	Possible Inter-Class Comparisons
CASIA-a	CASIA [V1]	624	1679	192,699
LEI-a	LEI	75	131	2646

The ‘Set Name’ column in Table 4.4 contains the names Masek assigned to his reduced subset of the full image datasets listed in the ‘Super Set’ column. Thus, the widely available CASIA V1 dataset was edited by Masek to remove any difficult images and renamed to CASIA-a. The same process was carried out by Masek for the LEI dataset, again renaming the new subset, this time to LEI-a.

The generic platform does not change the selection of images used to only include images that an algorithm is successful with, it uses the complete image dataset as supplied by the creators of that dataset. This explained the discrepancy in results, but in order to test Masek's code properly it should be tested with the CASIA-a dataset to ensure that the same results as Masek claimed can be achieved.

Masek states that the images used were “*those images that were segmented successfully*”, but does not identify them by name. With only the process to go on, the process was repeated. A simple check of CASIA V1 dataset images with Masek's segmentation boundaries overlaid was used to remove the most obvious segmentation failures. This is a manual process with the problems of accuracy, repeatability and bias inherent to such processes, but the first attempt reduced the number of images from 756 down to 633, only nine images more than the 624 used by Masek.

The criteria used for removal was any image where the overlaid outer iris segmentation boundary visibly did not follow the iris sclera boundary. The test results using this set of 633 images came close to the results achieved by Masek but were not quite as good. The process of removing images was repeated on the remaining 633 but this time looking at the iris pupil boundary. A further nine were removed to achieve the 624 image target, but the result attained using this dataset was still not the same as Masek's. The test results are detailed in Table 4.5.

Table 4.5 – Masek's claimed CASIA V1 results with generic platform (GP) results for the reduced 633 and 624 image CASIA V1 datasets and the full CASIA V1 results.

CASIA V1				
Method	Masek Thesis	Masek GP 624 Images	Masek GP 633 Images	Masek GP CASIA V1
True Match	-	100.0%	100.0%	98.0%
Separability	99.8%	96.5%	96.5%	61.1%
Decidability	6.2	5.4	4.7	3.6
Equal Error Rate (EER)	0.12%	0.10%	0.46%	4.1%
Average Code Creation Time (s)	-	6.41	6.39	6.46
Database Encoding Time (mins)	-	66	67	81
Database Analysis Time (mins)	-	61	62	71

Table 4.5 shows the results attained using Masek's code by Masek, by the generic platform using the 633 and 624 image subsets, and the last column showing the results using the full

CASIA V1 dataset. The decidability for the 624-image dataset is lower by 0.8 than Masek's results and the separability is lower by 3.3%, suggesting that the Hamming distances for the authentications and imposters are closer together and overlap more. Authentications is the term used by Daugman for those irises that correctly match to the right person and imposters is his term for those that are correctly rejected by the system.

The equal error rate of 0.1% actually improves on Masek's results and the true match result of 100% is the best that it can be. It is a shame that Masek did not provide a true match result with which this could be compared. The EER Hamming threshold gets closer to Masek's 0.4 with each set of tests but is still 0.02 higher in the end. The separability and decidability never quite match the results in the literature, but they come very close.

The tests were re-run a further ten times adding a different image from the removed images into the dataset each time but none of these tests got any closer than the two results in Table 4.5.

Although the testing was not able to match Masek's results, it did show how much better the results could be made by simply removing any images that the iris recognition algorithm could not cope with. The images that were removed to get to the 633 and 624 image datasets are stated in Appendix Appendix A. Figure 4.2 shows the distribution of Hamming distances for true matches versus false matches for Masek's iris recognition algorithm using the CASIA V1 dataset as tested within the generic platform.

The results achieved here provided significant confidence that, with the correct combination of images, the generic platform running Masek's algorithm could match the results that were presented in the literature.

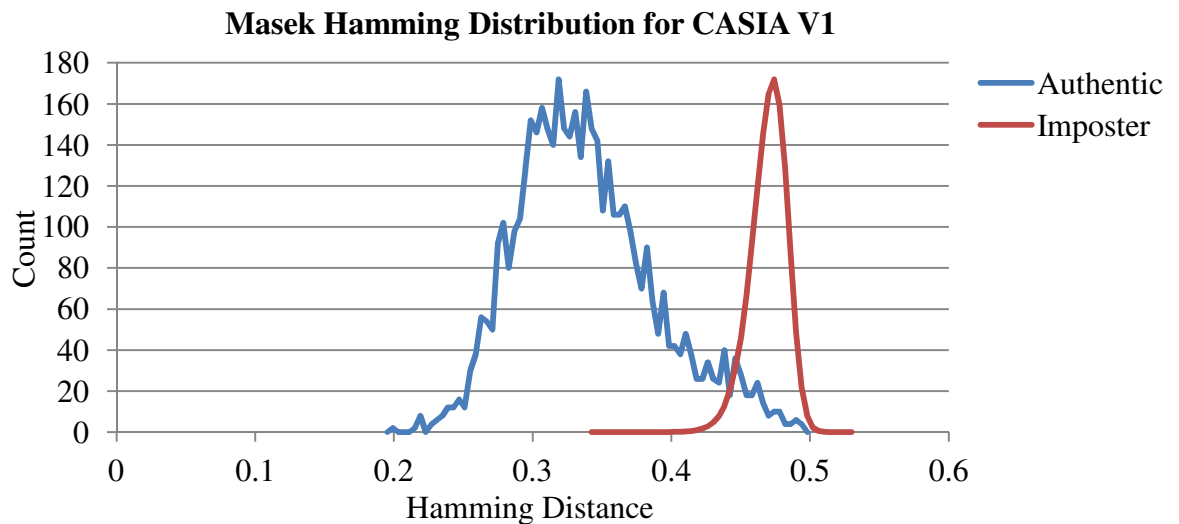


Figure 4.2 – The normalised Hamming distribution for Masek's flow using CASIA V1 produced using the generic platform

Ideally, the true matches (blue) in Figure 4.2 will be entirely to the left of the false matches (red). The more likely situation is that the blue and red will overlap each other, as seen here, but the smaller the overlap the better.

The two curves on Figure 4.2 are normalised by the generic platform in order to make the authentic curve readable. However, this normalisation does corrupt the point at which the authentic and imposter plots cross and alters how significant the overlap appears to be. Unfortunately, not normalising the curves renders the graph unusable.

4.1.2 Conclusions

The results using CASIA V1 appeared to be good, however, the only comparison available was Masek's own results and his results suggested much better performance than produced using the generic platform. Looking at the FAR against FRR curves produced from the data on page 43 of Masek's thesis (Figure 4.1) it would seem that Masek's algorithm is capable of achieving an equal error rate at or near zero.

The results from the generic platform in Table 4.3 show an equal error rate of 4.1%. The reasons for this were shown to be the result of careful selection of the dataset images that were used for testing. Selecting only the 'successfully' segmented images skewed Masek's results significantly, making the algorithm and code appear to function better than they do.

This is one of the reasons it is so important that there be a standard test and evaluation platform for iris recognition. The generic platform does not work out which images are segmented successfully and then use only those for testing. It evaluates the performance of an algorithm across the full range of images supplied, however successfully they segment.

The generic platform performs exhaustive one against all testing to establish the efficacy of each algorithm, then outputs a text file containing the results of the testing. Appendix A has an example of the contents of the text file.

The next section extends the testing of Masek's algorithm to the remaining seven datasets selected for testing and evaluates how it performs.

4.2 Further Dataset Evaluation with Masek's Algorithm

This section takes a more in-depth look at the results obtained using Masek's iris recognition flow. Masek's code is further evaluated with the remaining seven image datasets that were selected in Section 3.7.

Having tested Masek's code with the CASIA V1 dataset and arrived at a dataset that returned similar results to the CASIA-a dataset used in the literature, the code was then tested with each of the remaining seven image datasets.

The generic platform uses the selected algorithms to segment, normalise and encode each image in the selected image dataset, creating an iris code and mask for each image. When that is completed for all images, it uses the selected matching algorithm to compare the code for each image with the codes from all the other images in that dataset.

Masek's algorithms for segmentation, normalisation, encoding and matching were selected and run within the generic platform for each of the test datasets selected. These same datasets are used to test all the algorithms and their variations within this thesis, providing a consistent base with which to evaluate the performance of each algorithm and method. The recognition results from Masek's flow are shown in Table 4.6.

Table 4.6 – Results from Masek's algorithm with the remaining test datasets

Masek Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	98.0%	61.1%	3.6	4.1%
CASIA V2D1	0.6%	0.0%	-	35.7%
CASIA V2D2	0.3%	0.0%	-	25.9%
CASIA V3i	96.6%	35.2%	3.8	3.5%
IITD	0.3%	-	-	18.7%
MMU V1	48.1%	0.4%	1.2	29.6%
MMU V2	5.7%	0.0%	-	34.9%
WVU Free	1.3%	0.0%	-	43.9%
Average	31.4%	13.8%	1.1	24.5%
Variance	97.7%	61.1%	3.80	40.4%

The CASIA V1 and V3i results in Table 4.6 are close to 100% true match rates, the MMU V1 in third place only managed 48% and the rest are below 10% true match. This gives an average true match rate of 31.4%, which varies by 97.7%. The ideal results would be 100% average true match rate with 0% variance, so these are very poor results for a recognition algorithm.

The average separability of 13.8% combined with the average decidability of 1.1 and average EER of 24.5% also show an algorithm that struggles to separate authentications from imposters. The decidability cannot be calculated for five out of the eight datasets listed, due to the true and false match Hamming distances occupying the same range of values. Initially, the reason for this poor performance was thought to be due to Masek's segmentation algorithm being written for the CASIA V1 dataset.

A manual evaluation of the segmented images from the failing datasets showed that while the segmentation was not perfect, many images were being segmented properly, indicating that recognition rates should theoretically be higher. Nine of the images from the CASIA V2D1 dataset that were segmented by Masek's algorithm are shown in Figure 4.3.

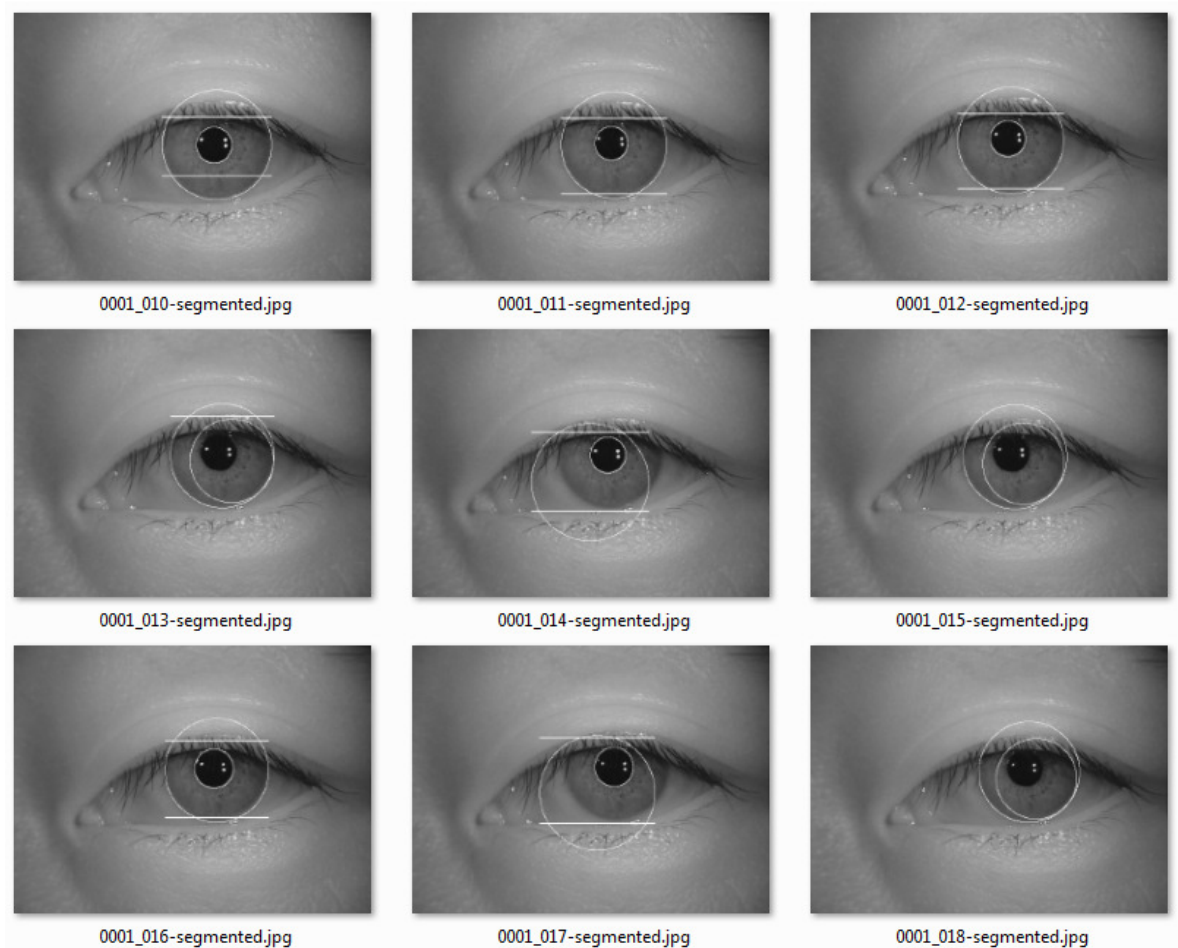


Figure 4.3 – Images from the CASIA V2D1 dataset that have been segmented by Masek's algorithm.

For the image results seen in Figure 4.3 Images 0001_010, 0001_011, 0001_012 and 0001_016 show the top eyelid, pupil and sclera iris boundaries correctly identified. Images 0001_013, 0001_015 and 0001_018 show failures in the pupil-iris and top eyelid boundary detection. Poor iris-sclera boundary detection can be seen in images 0001_014 and 0001_017. These segmentation results indicated that segmentation was better than the 0.6% true match rate in Table 4.6.

Since the segmentation results were better than expected, this suggested that the recognition problem might be later in the recognition process than the segmentation stage. By manually evaluating the normalisation process, the segmentation mask was identified as a possible source of the error.

The segmentation masking process involves marking any pixels on the original picture of the eye that are determined by the recognition algorithm as being 'noise'. Noise is defined

as anything that is *not* part of the iris, e.g. the pupil, sclera, eyelids, eyelashes, any specular reflections or other obstructions such as glasses. In the generic platform, this is a sub-function of the normalisation process.

For the CASIA V1 and V3i datasets, the segmentation mask output by Masek's code showed a clear iris, but the eyelashes and pupil were black instead of just dark grey. The source of this anomaly was a fixed value intensity threshold that was being applied to the images, setting all intensities below the threshold to zero. Figure 4.4a is image 001_1_1 from the CASIA V1 with the thresholding applied, Figure 4.4b is the same image without the thresholding.

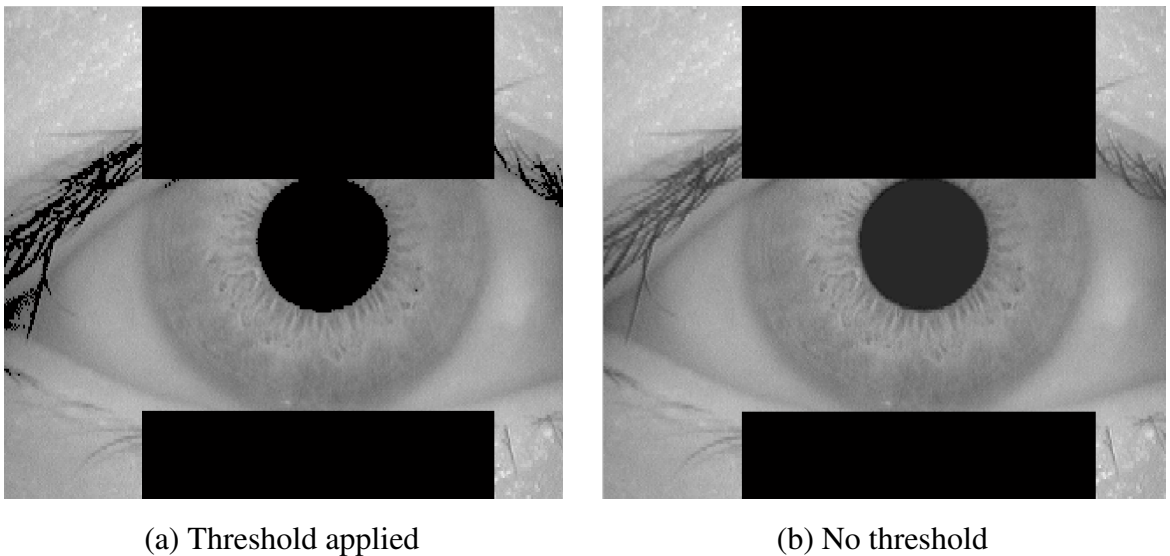
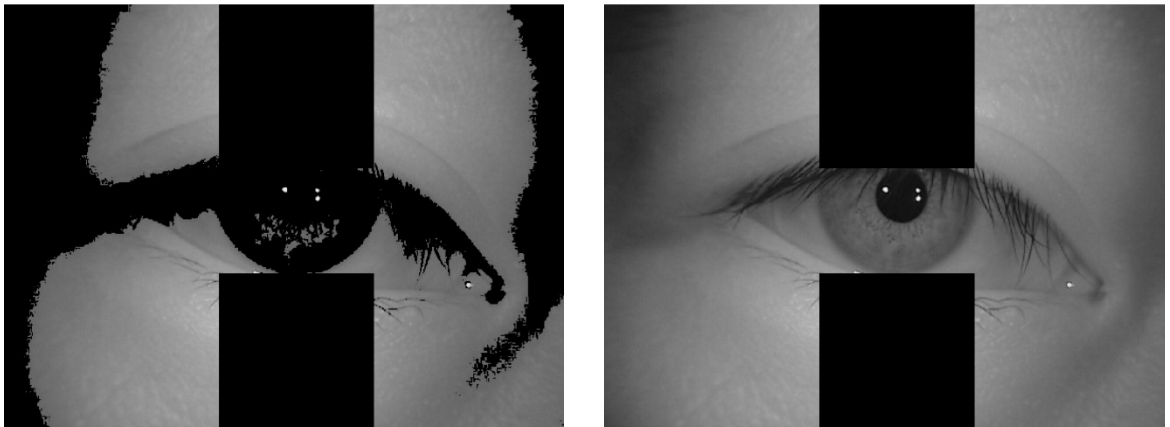


Figure 4.4 – Image 001_1_1 from CASIA V1 showing segmentation masking by Masek's algorithm (a) and with threshold applied (b) with no threshold applied

When the two images in Figure 4.4 are compared, the darker lashes of image Figure 4.4a show very clearly, it can also be seen that the pupil of Figure 4.4a is darker than that in Figure 4.4b. However, for the datasets with poorer results, such as the CASIA V2D1 dataset, large areas of the iris were blacked out.

Figure 4.5a is image 0000_10 from the CASIA V2D1 with the thresholding applied, Figure 4.5b is the same image without the thresholding.



(a) Threshold applied

(b) No threshold

Figure 4.5 – Image 0000_10 from CASIA V2D1 showing segmentation masking by Masek's algorithm (a) and with threshold applied (b) with no threshold applied

Comparing Figure 4.5a to Figure 4.5b the larger areas of black, caused by thresholding, in Figure 4.5a, especially over the iris, are very clear. In fact, nearly the entire iris in Figure 4.5a has been overwritten with black, as opposed to the clear iris region in Figure 4.5b.

The source of the blacked out areas seen in Figure 4.5a is the thresholding operation carried out at the end of Masek's segmentation mask code. This threshold operation sets all pixels of intensity 100 and lower to zero. For the CASIA V1 dataset this masking excludes the pupil and eyelashes from the final comparison. In Section 2.3 of Masek's thesis, he states:

“For isolating eyelashes in the CASIA database a simple thresholding technique was used, since analysis reveals that eyelashes are quite dark when compared with the rest of the eye image. Analysis of the LEI eye images shows that thresholding to detect eyelashes would not be successful. Although, the eyelashes are quite dark compared with the surrounding eyelid region, areas of the iris region are equally dark due to the imaging conditions.”

This added weight to the belief that the image thresholding was responsible for the poor performance in datasets other than CASIA V1.

Further analysis, looking purely at the unwrapped iris patterns and associated codes was carried out. Figure 4.6 shows the unwrapped iris codes and masks from image 001_1_1 shown in Figure 4.4; thresholded (a) and unthresholded (b).

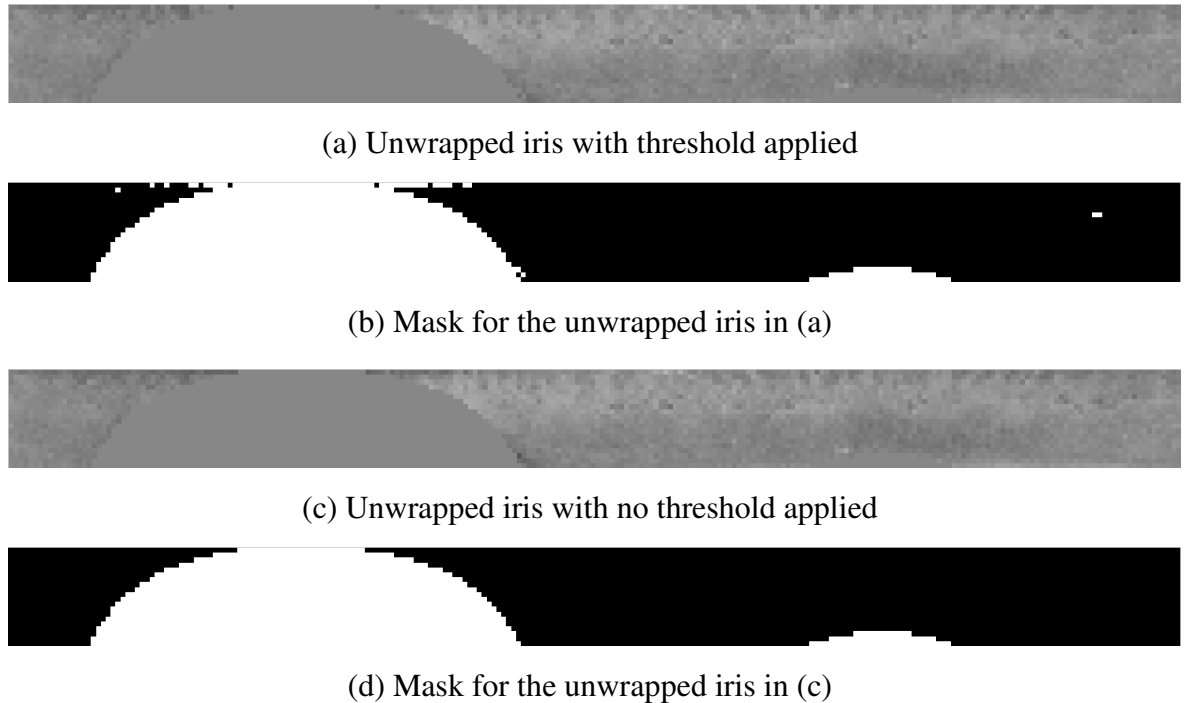


Figure 4.6 – unwrapped iris (a) and mask (b) of Figure 4.4a and unwrapped iris (c) and mask (d) of Figure 4.4b

The unwrapped iris patterns, (a) and (c), are almost indistinguishable from each other. The mask from the thresholded image (b) shows additional white markings when compared with (d), where white areas are the marked 'noise'. This indicates the detection of additional noise areas in the thresholded image.

Looking at the unwrapped iris and mask from the CASIA V2D1 image in Figure 4.7, (a) and (b) respectively, showed a very different picture. Figure 4.7 (a) and (b), respectively, show the unwrapped iris pattern and mask for Figure 4.5a.

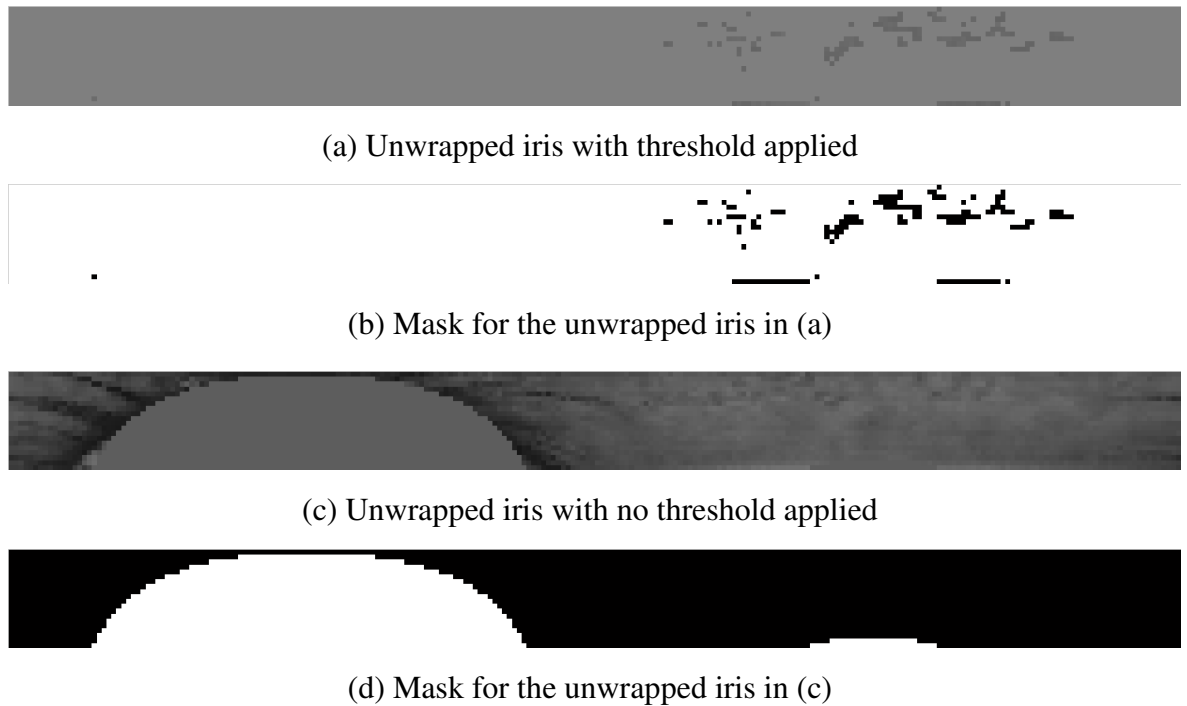


Figure 4.7 – unwrapped iris (a) and mask (b) of Figure 4.5a and unwrapped iris (c) and mask (d) of Figure 4.5b

The majority of both the unwrapped iris and the mask in Figure 4.7 (a) and (b) are a constant colour. The mask is mostly white, indicating mostly noise and the unwrapped iris is mostly a mid-level grey, also indicating noise.

Compare these to the unwrapped iris and mask from Figure 4.5 (b) where the thresholding was not applied. These are shown in Figure 4.7 (c) and (d). The unwrapped iris in Figure 4.7 (c) shows a semi-circle of flat grey, representing the top eyelid location, but the rest of the image is textured with a range of grey's. The mask shows the same information with a large white semi-circle for the top eyelid and a small white feature further along representing the bottom eyelid.

As the vast majority of iris code values in the unwrapped iris and mask in Figure 4.7 (a) and (b) are masked out, there is a much lower probability of useable iris code bits matching up during the matching process.

Given this thresholding problem, further investigations were carried out. First, performance was evaluated using Masek's algorithm with the segmentation mask thresholding removed.

Table 4.7 – Results of testing Masek's flow with the segmentation mask thresholding removed

Masek – No threshold				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	97.6%	51.7%	3.2	5.4%
CASIA V2D1	91.3%	64.3%	1.6	21.3%
CASIA V2D2	88.4%	53.0%	1.4	25.5%
CASIA V3i	96.7%	84.2%	3.7	4.0%
IITD	92.2%	-	2.9	9.1%
MMU V1	57.7%	33.8%	1.2	28.1%
MMU V2	44.0%	25.6%	0.9	35.1%
WVU Free	75.0%	48.0%	1.2	24.9%
Average	80.2%	18.3%	2.0	19.2%
Variance	53.6%	50.5%	2.7	31.1%

Compared with the results in Table 4.6, the results in Table 4.7 show a significant improvement in all except the CASIA V1 and V3i results, and the V3i results do improve slightly. An average true match rate of 80.2% is much improved over the 31.4% seen when the thresholding was applied. The average separability has increased by 4.5%, decidability has increased by 0.9 and EER is 5.3% lower indicating an improvement in the ability to separate authenticics from imposters. The significant decrease in the variances for all four metrics show that this algorithm is more consistent across the eight image datasets.

The results in Table 4.7 prompted further evaluation of the use of intensity thresholding during the segmentation mask stage. If the threshold of 100 used by Masek was so effective when used with the CASIA V1 dataset, and no threshold restored the effectiveness of his algorithm on the other datasets, then further investigation of values between 100 and nothing might produce improvements for the other datasets.

Three experiments were set up, the first using a threshold of 75, the second using a threshold of 50 and the third using a threshold of 25. Table 4.8 shows the results for using a threshold of 75.

Table 4.8 – Results of Masek's flow with a fixed segmentation mask threshold of 75

Masek – Threshold=75				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	98.3%	93.8%	3.4	3.9%
CASIA V2D1	1.8%	0.0%	NaN	26.7%
CASIA V2D2	0.7%	0.0%	NaN	31.3%
CASIA V3i	97.1%	62.7%	3.7	3.6%
IITD	53.0%	-	NaN	11.8%
MMU V1	14.3%	0.0%	0.6	40.4%
MMU V2	31.2%	0.0%	NaN	34.2%
WVU Free	71.3%	70.8%	1.3	22.5 %
Average	45.96%	28.4%	1.1	21.8%
Variance	97.6%	93.8%	3.7	36.8%

The results in Table 4.8 show a 14% improvement in average true match rate and small improvements in all the remaining averages and variances when compared to the results with a threshold of 100. It also showed a small improvement in the CASIA V1 results that was not expected, as Masek had tuned the threshold to the CASIA V1 and it was expected that this would be optimal.

While better than the results using a threshold of 100 these were nowhere near as good as using no threshold at all. The significant drop in the MMU V1 results are, however, quite concerning.

Table 4.9 – Results of Masek's flow with a fixed segmentation mask threshold of 50

Masek – Threshold=50				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	97.8%	94.8%	3.3	4.8%
CASIA V2D1	91.9%	30.5%	1.7	20.3%
CASIA V2D2	88.8%	67.3%	1.5	23.3%
CASIA V3i	96.8%	88.6%	3.6	3.9%
IITD	90.8%	-	2.9	9.6%
MMU V1	17.2%	0.0%	0.6	41.3%
MMU V2	45.0%	26.6%	0.9	33.4%
WVU Free	81.3%	60.7%	1.3	20.9%
Average	76.2%	46.1%	2.0	19.7%
Variance	80.6%	94.8%	3.0	37.4%

The results in Table 4.9 show a significant 45% improvement in average true match rate, a 25% improvement in average separability, an 84% improvement in decidability and a 4.9% reduction in EER when compared to the results with a threshold of 100. However, it also showed a small reduction in the CASIA V1 true match results.

Overall, this is a much healthier set of results as, with the notable exceptions of the MMU V1 and CASIA V1 results, the performance of Masek's algorithm has improved for all the other datasets, when compared to the same algorithm using a threshold of 100.

While significantly better on average than the results using a threshold of 100 or 75 these were not as good as using no threshold at all. The small reduction in the CASIA V1 dataset is insignificant when compared to the significant gains across most of the other datasets. However, the very poor MMU V1 results suggest that this is not a good threshold across the board.

Table 4.10 – Results of Masek's flow with a fixed segmentation mask threshold of 25

Masek – Threshold=25				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	97.5%	51.4%	3.2	5.1%
CASIA V2D1	92.5%	66.4%	1.6	21.3%
CASIA V2D2	89.6%	52.0%	1.4	25.1%
CASIA V3i	96.8%	89.3%	3.6	4.0%
IITD	91.7%	-	2.9	9.0%
MMU V1	18.7%	0.0%	0.5	41.9%
MMU V2	44.8%	16.3%	0.9	34.0%
WVU Free	78.8%	57.1%	1.3	20.1%
Average	76.3%	41.6%	1.9	20.1%
Variance	78.8%	89.3%	3.1	37.9%

The results in Table 4.10 show very little change from using a threshold of 50 when compared to the results with a threshold of 100. Another small reduction in the CASIA V1 true match results is offset by a small increase in the MMU V1 results.

Overall, the results from using a threshold of 25 are very slightly worse than those using a threshold of 50, and still not as good as using no threshold at all.

The results across the datasets show conclusively that, using a fixed pixel intensity threshold of 100 is at best sub-optimal, and at worst it destroys any real chance of meaningful iris recognition. What is more interesting is the intensity threshold levels at which recognition performance for each dataset peaks.

Recognition performance for the CASIA V1 and V3i datasets both peak at a threshold of 75, but the performance over varying threshold values is almost constant, so this is not a significant result. For the remaining datasets, except MMU V1, peak recognition performance occurs at a threshold of 50. For the MMU V1 dataset this peak does not occur

until there is no threshold applied and recognition performance across all other datasets is very close to its best if no threshold is applied.

Figure 4.8 shows this information graphically, highlighting the huge improvement in most of the recognition results as the threshold is moved down to 50. The black line is the average.

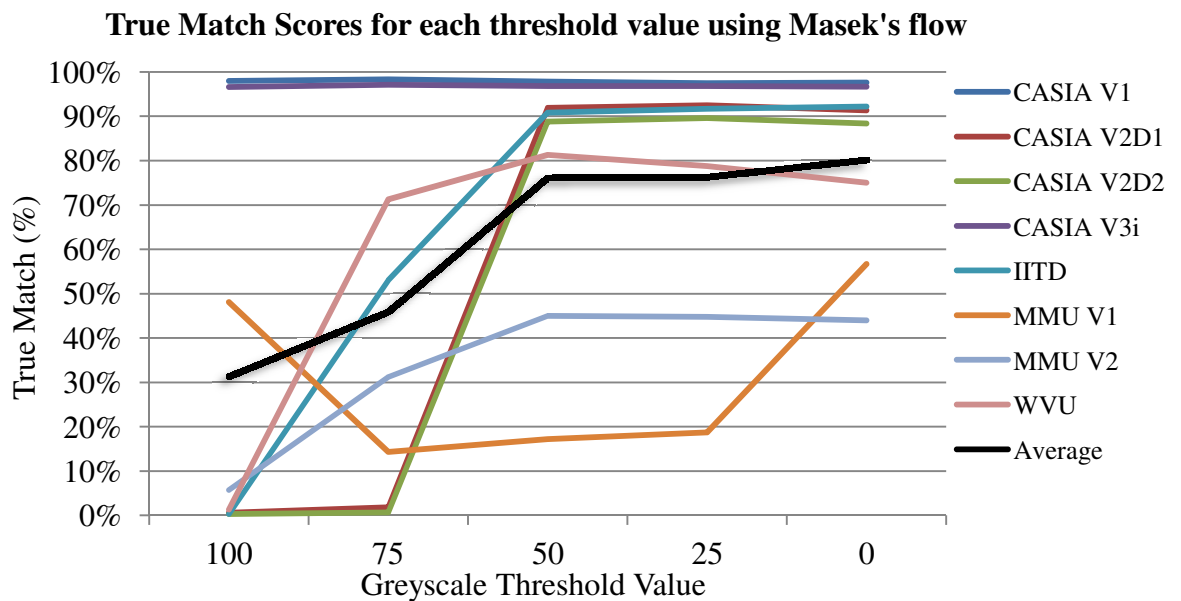


Figure 4.8 – True match scores for each image dataset when used with Masek's algorithm using a range of threshold values

Figure 4.8 clearly shows the minimal reduction in recognition rates for reducing the threshold below 50. The minimal variation in the CASIA V1 and V3i results, combined with the minimal reduction in recognition rates for those datasets where rates peaked at 50, suggests that overall, thresholding is having only a small positive impact.

Setting the threshold too high for a given dataset destroys the ability for an algorithm to achieve useable recognition rates. Removing the threshold altogether may fractionally reduce some datasets recognition performance, but across a wide selection of datasets the highest overall performance is achieved by not performing intensity thresholding.

The black average curve on Figure 4.8 illustrates this point very clearly by showing the overall improvement in true match rates despite small reductions for four of the datasets. As the intensity threshold value falls, the average true match percentage rises. This also

highlights the need to have a good cross-section of datasets as, without the MMU V1 dataset, the conclusion here would have recommended a fixed threshold of 50.

The threshold is used to modify the mask so that it excludes eyelash “noise”. Unfortunately, the use of a static threshold results in a mask that is imperfect and that’s what causes the trouble in this situation. Masking the eyelashes and other noise such as specular reflections correctly can lead to an improvement in recognition rates but a more adaptable method is needed [140, 141, 142].

The other aspect of using the platform for iris recognition, is the time it takes to perform each step of the process and the overall process time. Table 4.11 shows typical processing times for the segmentation, normalisation and encoding process, summed up as the iris code creation time. Overall times for the code creation and analysis of each dataset when using Masek’s algorithm within the generic platform are also shown.

Table 4.11 – Processing times from using Masek’s flow without thresholding. (s) seconds, (h:mm) hours and minutes.

Masek Processing Times			
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)	Analysis Time (h:mm)
CASIA V1	6.5	1:21	1:10
CASIA V2D1	24.4	8:07	2:52
CASIA V2D2	22.3	7:26	2:36
CASIA V3i	9.8	11:17	13:38
IITD	10.6	6:34	11:33
MMU V1	4.5	0:33	0:23
MMU V2	6.0	2:01	1:53
WVU Free	15.9	0:21	0:01
Total		37:41	34:05

The analysis time column in Table 4.11 contains the time taken for the generic platform to do a full one-against-all comparison of all the encoded iris codes for each image dataset. If

every image in an n-member dataset is compared against all the other images, there will be a total of $\frac{n(n-1)}{2}$ comparisons, so the time taken can be expected to scale with n in this, quadratic, manner.

Figure 4.9 is a scatter diagram showing the analysis time against the number of images in the dataset used.

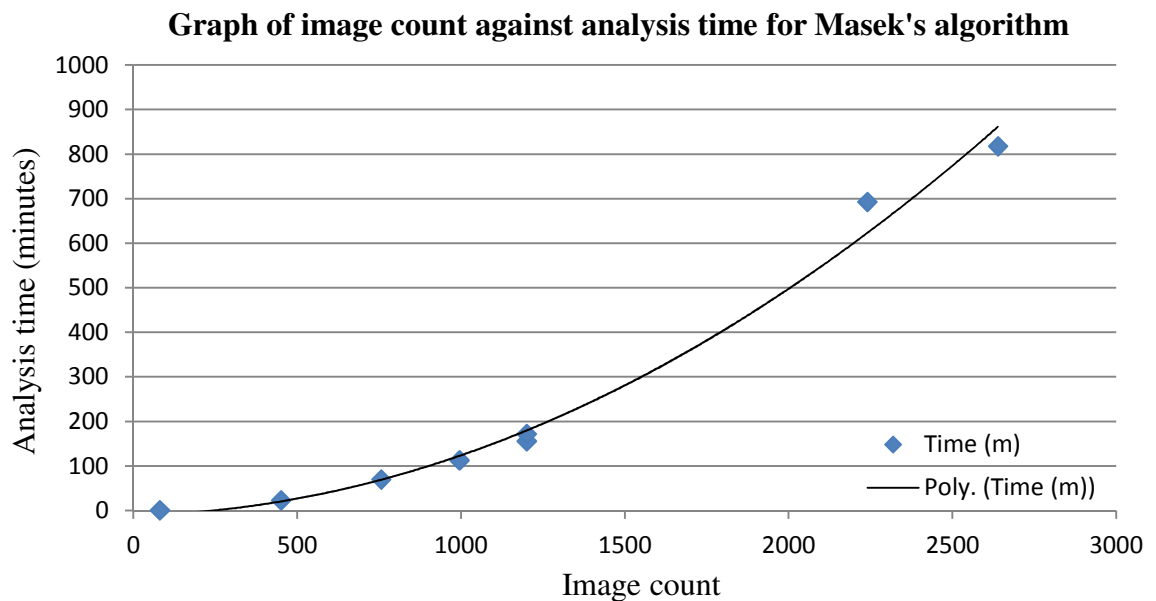


Figure 4.9 – Graph plotting the number of images in a dataset against the time taken to analyse the recognition performance in minutes using Masek's matching algorithm within the generic platform

Figure 4.9 shows the analysis times for each dataset using Masek's matching algorithm within the generic platform. The points of the scatter diagram follow the expected quadratic polynomial pattern, with minor variation caused by system load changes. This shows that the analysis time is derived directly from a quadratic polynomial function of the image count and is independent of the image size with no significant deviation introduced by the generic platform.

The code creation time in Table 4.11 is the total time, rounded to the nearest minute, that the generic platform took to convert all of the images in each dataset into code and mask pairs using Masek's segmentation, normalisation and encoding algorithms. The average code creation time is the average time in seconds taken to segment, normalise and encode each individual image in the dataset.

While at first sight the code creation times appear to show a correlation between image size and processing time, on further investigation it is not that simple. Figure 4.10 is a scatter diagram of the average time taken by Masek's iris recognition algorithm to create the iris code for images in each dataset plotted against the number of pixels in each image

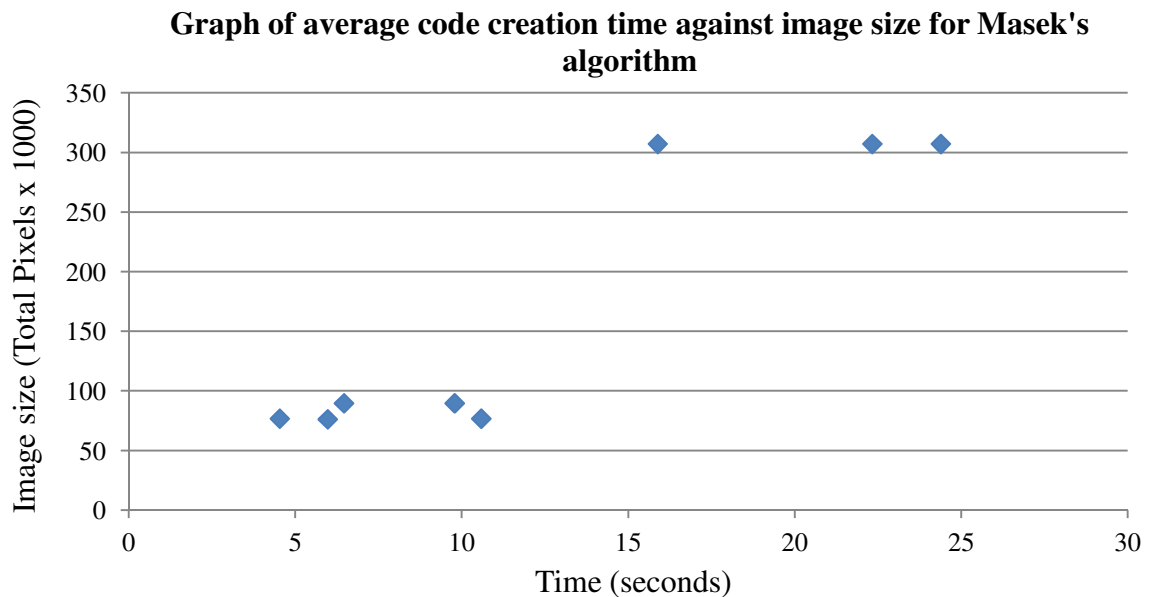


Figure 4.10 – Graph plotting the number of pixels in images in each dataset against the average time in seconds taken to create the iris code for images in the dataset using Masek's matching algorithm within the generic platform

Figure 4.10 shows that there is a correlation between image size and code creation time for Masek's algorithm, as the much bigger 640×480 pixel images do take the longest. However, there is no direct relationship shown between image size and iris code creation time.

This lack of a direct relationship between processing time and image size is because the most significant portion of the code creation time for Masek's algorithms is the circular Hough transform [41] used for the iris-sclera boundary detection.

For the iris-sclera boundary, the Hough code looks for circles ranging in size between 80 pixels and 150 pixels. For a 320×280 image, this gives a Hough accumulator that is 320×240×71 (5,452,800 bytes). For a 640×480 image the accumulator will be 640×480×71, or 21,811,200 bytes. Therefore, a fourfold increase in image size gives a fourfold increase in accumulator data to process.

However, the circular Hough transform uses an edge image of the input iris image, generated using a Canny edge detector [45], to determine the potential boundary points for the sclera-pupil boundary. One circle is drawn at each circle radius in the selected range (80 to 150) for every single point in the edge image. The number of points in the edge image is determined by the thresholds set for the Canny edge detector, the lighting at the time the image was captured and the details present within the image.

Unfortunately, this means that the number of points in the edge image is not consistent across images or datasets. Thus, it is possible for the Canny edge image of a 640×480 image to have fewer edge points than the edge image for a 320×240 image.

This variation in edge points leads to a significant variation in the number of circles drawn by the circular Hough transform, which leads to a significant variation in the time taken to create the iris code for each image.

Thus, the execution time for Masek's algorithm is dependent upon the size of the image as well as the number of points that are returned in its edge image. The bigger the image the more data there is to process, but the more points that are present in the edge image the more processing has to be carried out. For this reason it is not possible to normalise these results in any meaningful way to remove the effect of the image size.

4.2.1 Conclusions

This section looked at the work carried out when evaluating Masek's algorithm on all the image datasets selected for testing the generic platform. It showed the initial results and how they led to an investigation of why those initial results were so inconsistent.

It was found that the application of a fixed threshold to the images to remove 'noise' was the reason for these inconsistent results, so a general investigation into the effect of thresholding on recognition results was carried out.

It was very clear from the threshold test results that using a fixed threshold value with Masek's flow does have a negative effect on the results when using all of the datasets except the CASIA V1 and V3i. Without the threshold, Masek's flow performs far more competently across the all eight of selected datasets, as seen in Table 4.7.

Since Masek's flow was being used as the baseline, and parts of Masek's flow were used when testing other recognition functions when they were not themselves 'fixed flow's, the fixed threshold was removed, so that it did not adversely affect recognition performance. This also meant any results comparisons would be against the results obtained from using Masek's flow without thresholding.

However, it was preferable to keep Masek's code as close to the released code as possible, thus continuing to allow testing via the GUI to use the original release code, so Segmentation Mask One (SM1) was created, Section 5.7. SM1 copes with lines, circles and ellipses for all four boundaries, does not threshold in any manner, and no effect of doing this could be measured on any of the datasets when compared to Masek's segmentation mask routine when used with Masek's normalisation algorithm. Thus, SM1 was used as the default segmentation mask routine for all other testing.

Evaluating the performance across datasets, the WVU Free and CASIA V2 datasets are the slowest but also have the biggest images. At a quarter of the size, the CASIA V1, and MMU datasets are much faster to encode.

Within the times there are two anomalies, both the IITD and CASIA V3i datasets are much slower than the other datasets of the same size. The majority of this time was in the Hough transforms in the segmentation stage, so this just indicated that the edge images contained

more detail, meaning that the Hough transforms had more circles to draw. This may mean that the edge detection thresholds may not be optimal for these datasets.

All of this information clearly shows the advantage of using the platform for testing. Comparisons across datasets become a simple matter of time, very little effort is required to set up the tests, and once started the platform can be left alone to get on with the job. The information returned was very helpful in evaluating the performance of the algorithms under test and tracking down any problems.

The next section adds other pre-existing open-source iris recognition flows that were written for MATLAB to the generic platform. It then compares them to the results obtained with Masek's flow without thresholding to evaluate their performance.

4.3 Adding Other Open-source Iris Recognition Algorithms

This section discusses the process of adding pre-existing open-source iris recognition algorithms to the generic platform, and the results obtained from testing them there.

As discussed in Section 2.7, there were three open-source iris recognition algorithms written in MATLAB. Masek's had already been integrated into the generic platform, but the remaining two, introduced in Section 2.7, from Antonino [38] and Li [12, 27] had not. Antonino's algorithm is a replacement for the encoding and matching routines provided by Masek, while Li provides an entire iris recognition flow and had published the work demonstrated by his algorithm.

There were two open-source iris recognition algorithms written in a programming language other than MATLAB. The first of these was JIRRM [40] a Java based GUI that was of interest because, like Masek's code, it also used the Hough transform [41] for the pupil and iris segmentation. This offered a potential contrast with Masek's algorithm as they both used the same fundamental methodology to determine the boundaries, but there were algorithmic differences. JIRRM also implemented the normalisation algorithm with a simple Daugman style unwrapping operation [76].

The other open-source algorithm was OSIRIS [110, 39], a C/C++ based iris recognition algorithm with basic analysis functionality built-in. This was of interest because its methodology was very different from that used by any of the other algorithms tested and it was also written in C/C++, meaning that if a full conversion could be carried out then comparative timings could be made between C/C++ and MATLAB implementations of the same algorithm. OSIRIS was also developed by a research group and had been used for published works.

The first section looks at Antonino's encoding and matching algorithms.

4.3.1 Antonino's Encoding and Matching Code

'IRIS-ICT' [38] by Antonino is a modification of Masek's work and is written using MATLAB 7.3. The main script *irisRecognition* accepts two greyscale images as its input, though RGB images can be used by modifying the script *estraiIride*. It has the same key features of ease of use and accessibility as Masek's code.

This script uses Masek's segmentation and normalisation code and is supplied with copies of both. Encoding and matching are where the algorithms differ, as Antonino uses the inbuilt 2D MATLAB wavelet packet decomposition command, `wpdec2`, for encoding. The system outputs the Hamming distance of the degree of similarity between the two input images.

The matching routine calculates the "vettore energie", or energy carriers for the normalised iris which are then used to select elements of the wavelet packet tree, produced during encoding, which are then compared and averaged to get the final Hamming distance [100].

Antonino's algorithms are unusual in that, instead of creating a 2048-bit code and mask like Daugman [76], Masek [29] and many others [87, 63, 134, 12, 30, 99, 102], Antonino's encoding algorithm stores a text report generated by the `wpdec2` command. The Hamming distance taken in his matching code is the difference between the text files generated for each eye.

The disadvantage of this text-based iris code is that it is impossible to adjust the two iris codes to compensate for rotational variation between images of the same eye. There were no obvious advantages but it was considered that this might be a faster matching algorithm than Masek's.

Antonino's code was integrated into the generic platform as both a 'fixed flow', where there is no ability to select functions, other than matching, and as a 'custom flow' where the user can independently select functions and sub-functions.

Table 4.12 shows the recognition results achieved by Antonino's flow. Since Antonino uses Masek's segmentation and normalisation, the thresholding was also present. This thresholding was removed for consistency, as discussed in Section 4.2.

Table 4.12 – Results of Antonino's flow without a fixed segmentation mask threshold.

Antonino's Flow				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	64.3%	3.7%	1.2	28.8%
CASIA V2D1	85.4%	8.9%	1.1	31.2%
CASIA V2D2	78.3%	7.1%	1.0	34.0%
CASIA V3i	77.3%	5.4%	1.4	23.4%
IITD	78.2%	-	2.1	14.9%
MMU V1	13.0%	0.1%	0.4	41.4%
MMU V2	31.8%	2.2%	0.7	36.5%
WVU Free	36.3%	7.5%	0.6	40.9%
Average	58.1%	5.0%	1.1	31.4%
Variance	72.4%	8.8%	1.7	26.5%

The results in Table 4.12 show mediocre to poor true match results for Antonino's algorithm with 60.3% to 85% for the easier CASIA and IITD datasets and only 13% to 36.3% for the more difficult MMU and WVU datasets. No results have been published for this algorithm so there are no results with which to compare the performance achieved within the generic platform.

The average separability of 5% combined with an average decidability of 1.1 and average EER of 31.4% indicate that this algorithm struggles to separate authentications from imposters, confirming its mediocre performance. The low variances in separability and decidability indicate that this inability to separate imposters from authentications is consistent across the eight images datasets used.

Figure 4.11 shows the true match results from Table 4.12 compared with those obtained using Masek's algorithm with no thresholding applied, Table 4.7.

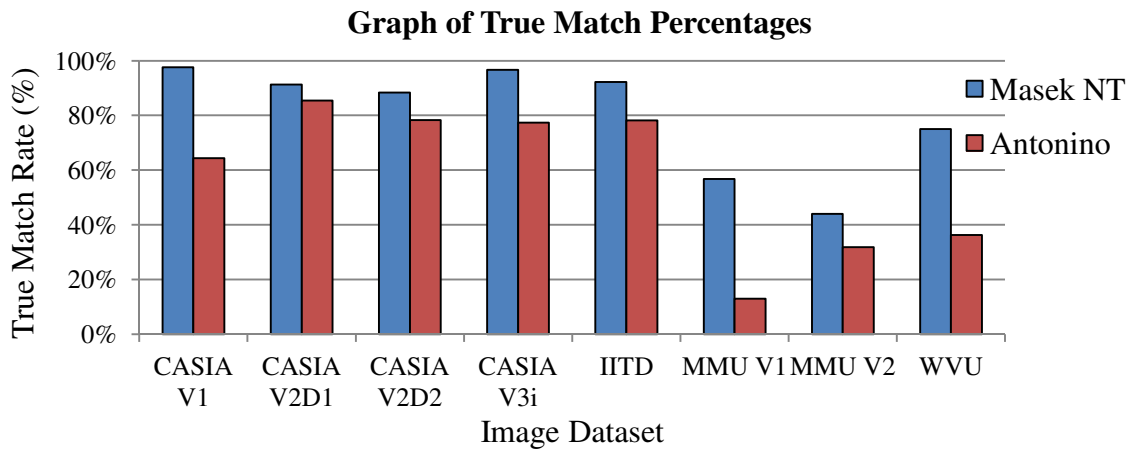


Figure 4.11 – Comparison of true match results from Masek's algorithm with no thresholding (NT) against the true match results attained by Antonino's algorithm.

Segmentation Mask One and Masek's segmentation and normalisation algorithms were used on both sets of results shown in Figure 4.11. The CASIA V2D1 result using Antonino's algorithm is only 5.9% lower than the result for Masek's algorithm for the same dataset. However, the remaining true match results are all between 10.1% and 43.7% worse when using Antonino's encoding and matching algorithms in place of Masek's encoding and matching algorithms.

Antonino's iris recognition code has two shortcomings that are responsible for reducing its efficacy. The first of these is that the encoding and matching processes completely ignore the noise mask and so they include areas of the unwrapped iris that are unlikely to match between images of the same eye, such as eyelashes and eyelids.

Masek's normalisation algorithm sets these regions to a mid-range grey, the average intensity for the specific image, in order to remove discontinuities for the encoding process. The replaced pixels will vary from image to image in both location and intensity, thus they will reduce the matching ability of the algorithm unless they are masked out. There is also no way to use the mask without completely changing the algorithm.

The other problem with Antonio's algorithm is the Hamming distance code. Eyes rotate within the socket and the angle at which an image is captured with respect to the rotation

of the eye changes between images. Rathgeb *et al.* [28] showed that correctly rotating iris codes improves performance, thus, it is reasonable to postulate that not rotating them will result in a less effective algorithm.

Antonino's code is not able to rotate the iris codes to find a better match and there is no minor change that would allow this to happen. The use of a the text results report means that all positional information is lost in the iris code.

The results in Table 4.13 shows the processing times for Antonino's flow with the various test datasets.

Table 4.13 – Processing times from using Antonino's flow without thresholding. (s) seconds, (h:mm) hours and minutes.

Antonino Processing Times			
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)	Analysis Time (h:mm)
CASIA V1	6.5	1:15	1:10
CASIA V2D1	24.4	8:07	2:45
CASIA V2D2	22.3	7:28	3:13
CASIA V3i	9.8	11:12	9:23
IITD	10.7	6:38	6:19
MMU V1	4.5	0:33	0:25
MMU V2	6.0	2:01	1:54
WVU Free	15.9	0:21	0:01
Total		37:35	25:10

Since the segmentation and normalisation algorithms are Masek's code, and the encoding process is a very small percentage of the time taken to create and iris code from an image, it was expected that the code creation times would be similar to Masek's. Table 4.13 confirms that this is the case, with small variations to the times achieved with Masek's flow, but no large differences.

Therefore, the pertinent results in Table 4.13 are the analysis times where Antonino's code had the potential to be faster. These times are compared directly with the times achieved with Masek's algorithm in Figure 4.12.

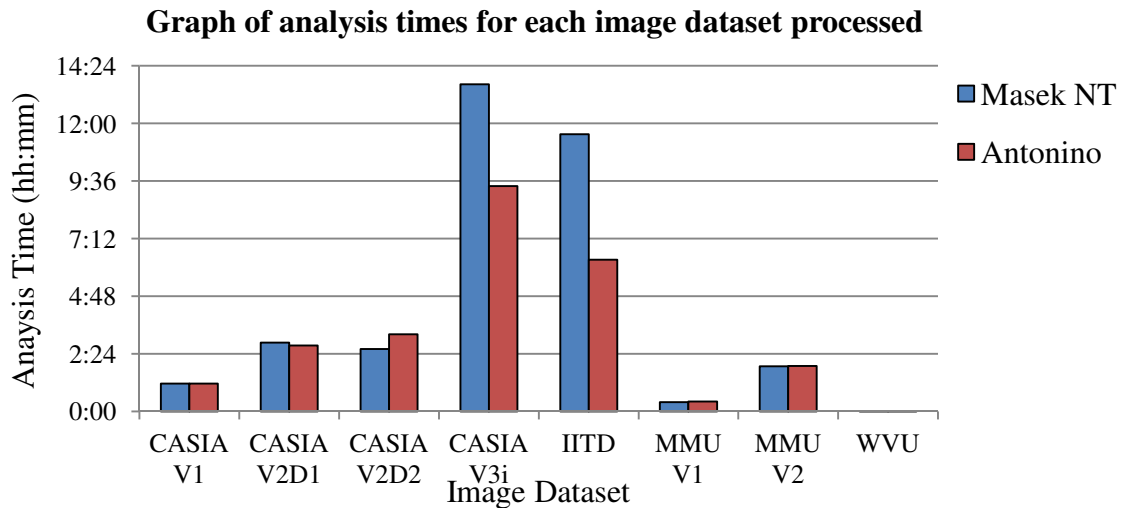


Figure 4.12 – Comparison of analysis times from Masek's algorithm with no thresholding (NT) against the analysis times attained by Antonino's algorithm.

Figure 4.12 did not reveal the anticipated consistent increase in performance over Masek's results. The CASIA V3i and IITD datasets do show a marked improvement over the times obtained using Masek's algorithm but the CASIA V2D2 and MMU V2 show a slight increase in analysis times. The remaining datasets show very similar analysis performance.

The CASIA V3i and IITD datasets tending to produce simpler codes, so requiring less computation to process them which enabled the total analysis time to be nine hours less than with Masek's algorithm, approximately 33% less time. Had this been consistent across the datasets this would have been a more interesting result.

Antonino has investigated a different way of matching iris codes, but the inclusion of noise in the matching process and the inability to rotate the iris templates to find the best match mean that this method is ultimately flawed, and it is unlikely that further work on it would improve it.

The next section looks at the open-source iris recognition algorithm written by Li, this was also written for MATALB.

4.3.2 Li's Iris Recognition Code

'Iris' by Li [12, 27] is a script to evaluate his work on iris recognition of non-ideal images. The script is supplied with an 80-image subset of the full 800-image WVU dataset and is written in MATLAB.

The key aims of Xin Li's scripts are to demonstrate his research work and to provide other researchers with the ability to analyse and build on that work. The fact that it uses MATLAB simply adds to its accessibility.

The purpose of Li's work was to explore non-ideal iris images, and his code is a 'fixed flow' from segmentation to matching. The WVU Free dataset that was used for the development of this algorithm contains a great variety of iris images, from ideal to non-ideal off-angle images.

Li's code aims to demonstrate the performance of his algorithm and so is more integrated than that in Masek or Antonino's work. The top level file attempts more than just encoding and matching a single iris image, it aims to encode and analyse the results across the entire image dataset provided, and also to output graphs of those analyses.

Li uses Canny edge detection with ellipse fitting for segmenting the image. Normalisation is carried out in the standard Daugman manner of polar unwrapping of the iris, but using ellipses instead of circles. Iris encoding uses a mean squares function with matching achieved using a correlation filter algorithm [109, 105].

The different encoding output means that a standard Hamming matching routine cannot be used. The correlation filter based matching produces a very different result to the Hamming distance, though it is still a single number. The correlation matching method returns a higher number, for two images that match more closely. Numbers as high as 40 were seen during testing.

Unfortunately, this code, as released, is not functional. This is because of basic, but fundamental, errors in the code, so it required significant debugging before the algorithm would run.

Once debugged, these results were obtained with the WVU Free image dataset provided:

Average Segmentation time:	1.41 seconds
Average normalisation & encode time:	0.65 seconds
Average intra-class distance time:	0.03 seconds
Average inter-class distance time:	0.91 seconds

These times were measured using the MATLAB platform and are averages over all 80 images in the WVU Free dataset.

Li's script is designed to perform a full evaluation of the image set provided. During the process images are output to defined folders showing the results of each stage. Figure 4.13 shows the segmentation lines on images 1_0 and 1_2 from the WVU Free as output by the script.

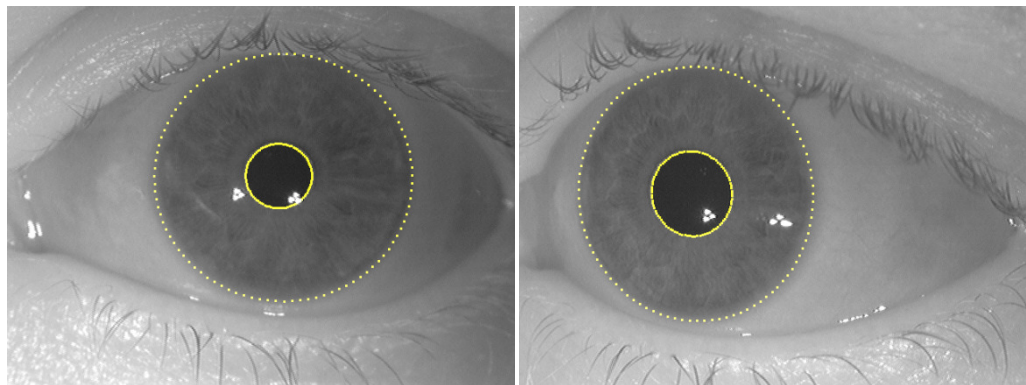


Figure 4.13 – Images 1_0 (left) and 1_2 (right) from WVU Free image dataset, the yellow ellipses indicate the detected iris boundaries

The images in Figure 4.13 are both of the same eye, but looking in different directions. The segmentation boundaries showing Li's use of ellipses instead of circles for the iris boundaries, is especially clear on the right hand image (1_2).

The very significant number of bugs in the code as provided, and the lack of documentation make it very difficult to access Li's code without significant understanding of both MATLAB and the underlying principles of iris recognition. Since Li designed the algorithm to work with the WVU Free dataset, his code should be at least able to process this dataset within the generic platform, so that was the point at which the error correction was stopped.

Unfortunately, even after debugging, only the CASIA V1 and WVU Free datasets were able to complete with Li's code, so results cannot be given for the other datasets. This inability to complete was due to the ellipse code being unable to find ellipses for all the images. While attempts were made to correct this by manipulating thresholds used in the Canny edge detection, these changes were not successful.

Li uses a pseudo random number generator in the correlation filter matching routine, meaning that different results were returned for each execution of the algorithm. No information on the reasoning behind the use of random numbers in the correlation filter was found in the written works, but it is possible that it was added to simulate the geometric calibration discussed in Li's paper [12]. Attempts were made to contact Li about the use of pseudo random numbers in his algorithm, but no response was received.

Table 4.14 – The results obtained over ten runs when using Li's flow with the CASIA V1 image dataset

CASIA V1				
Test	True Match	Separability	Decidability	Equal Error Rate (EER)
1	12.6%	0.0%	0.4	58.0%
2	12.4%	0.0%	0.4	58.0%
3	12.6%	0.0%	0.4	58.3%
4	12.8%	0.0%	0.4	58.0%
5	12.2%	0.0%	0.4	58.2%
6	12.6%	0.0%	0.4	58.0%
7	13.5%	0.0%	0.4	58.2%
8	12.7%	0.0%	0.4	58.1%
9	11.9%	0.0%	0.4	58.1%
10	12.2%	0.0%	0.4	58.2%
Average	12.6%	0.0%	0.4	58.1%
Variance	1.6%	0.0%	0.0	0.2%

The results in Table 4.14 show an algorithm that does not usefully recognise iris images correctly. The variance in true match rates of 1.6% shows that small differences in the correlation filter parameters can make a measurable difference, but it has no impact on the usefulness of this algorithm. The variance of 0.23% in the equal error rate (EER) also has no impact on the usefulness of this algorithm. Both the separability and decidability return consistently low results of 0.0% and 0.4 respectively. This indicates that the algorithm is unable to separate the authentications from the imposters.

Table 4.15 – The results obtained over ten runs when using Li’s flow with the WVU Free dataset

WVU Free				
Test	True Match	Separability	Decidability	Equal Error Rate (EER)
1	56.3%	0.0%	0.4	52.3%
2	56.3%	0.0%	0.4	52.1%
3	57.5%	0.0%	0.4	52.9%
4	56.3%	0.0%	0.4	52.3%
5	56.3%	0.0%	0.4	52.5%
6	57.5%	0.0%	0.4	52.9%
7	57.5%	0.0%	0.4	52.9%
8	56.3%	0.0%	0.4	52.5%
9	57.5%	0.0%	0.4	52.9%
10	57.5%	0.0%	0.4	52.1%
Average	56.9%	0.0%	0.4	52.5%
Variance	1.2%	0.0%	0.0	0.8%

The true match results are much better for the WVU dataset but still very poor. As with the CASIA V1 results, the true match variance of 1.2% is not enough to affect significantly the results obtained, neither is the EER variance of 0.8%. As for the CASIA V1 dataset, the separability and decidability return very low results of 0.0% and 0.4 respectively. This again indicates that the algorithm is unable to separate the authentications from the imposters.

While more runs were considered, the variance between runs seen over the first ten cycles was so small that further testing was considered unnecessary.

Table 4.16 summarises the average results from running Li’s flow with the CASIA V1 and WVU Free test datasets.

Table 4.16 – Average results of ten runs of Li’s flow with the test datasets within the generic platform from Table 4.14 and Table 4.15

Li’s Flow				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	12.6%	0.0%	0.4	58.1%
WVU Free	56.9%	0.0%	0.4	52.5%
Average	34.5%	0.0%	0.4	55.5%
Variance	43.7%	0.0%	0.4	5.1%

As stated already, the averaged results from Table 4.14 and Table 4.15 that are presented in Table 4.16, show a poorly performing iris recognition algorithm. The average across the datasets is also very poor at 34.5%. The results in Table 4.16 are contrasted in Figure 4.14 with those obtained with Masek and Antonino’s results without thresholding.

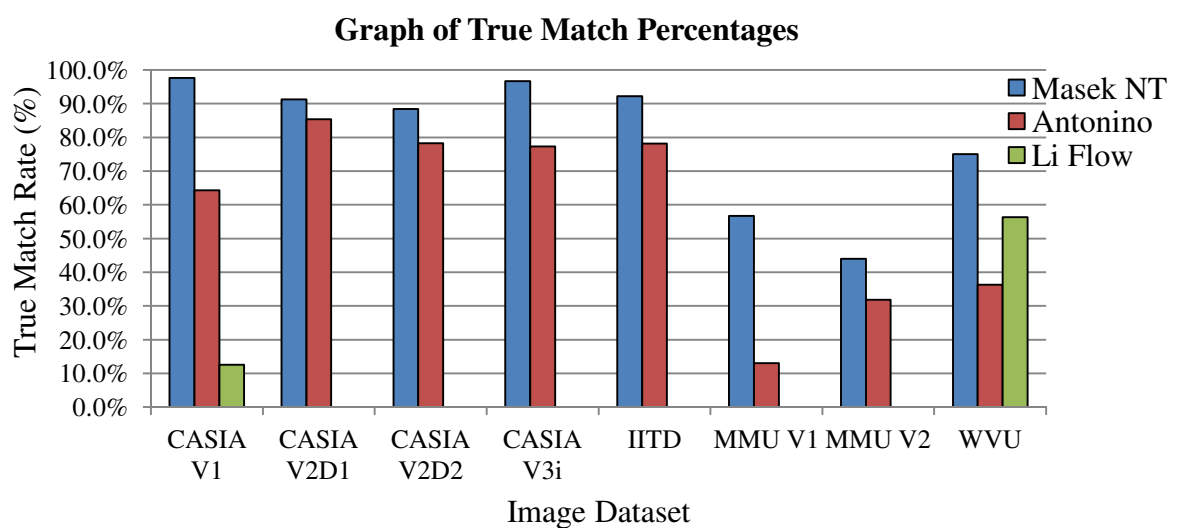


Figure 4.14 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino and Li’s algorithms.

The results in Figure 4.14 demonstrate just how poorly this algorithm performs on the CASIA V1 image dataset when compared to Masek and Antonino's algorithms. However, the WVU Free result compares much better, sitting centrally between Masek's 75% and Antonino's 36.3%. Considering that Li's algorithm was designed to work with the WVU Free dataset, the results are not as good as was initially anticipated. One advantage that Li's algorithm does have though, is in speed.

Table 4.17 – Processing times from using Li's flow for the two datasets that were completed. (s) seconds, (h:mm) hours and minutes

Li Processing Times			
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)	Analysis Time (h:mm)
CASIA V1	1.06	0:14	0:21
WVU Free	1.6	0:02	0:01
Total		0:16	0:22

The performance results in Table 4.17 were significantly faster than anything from Masek or Antonino showing that code creation and analysis times in MATLAB do not need to be measured in tens of seconds.

With such poor true match results there was merit in evaluating whether it was Li's whole flow that was problematic, or just parts of it. Two additional tests were run with all the datasets, the first using Li's segmentation and normalisation, and the other using Li's segmentation with Normalise One, both of these using Masek's encoding and matching algorithms.

Normalise One is described in full in Section 5.5, and replaces Masek's normalisation routine with one that had no measurable effect on the results output by the generic platform for the same input data, but can also handle ellipses. Normalise One also uses Segmentation Mask One as its default segmentation mask. The pseudo random number generation is only present in Li's matching routine and is therefore not included in the code for the remaining tests, so only one run was necessary.

Unfortunately, the ellipse detection problem which prevented results being obtained for the remaining six datasets was present in Li's segmentation algorithm, so it was again only possible to run these tests on the CASIA V1 and WVU Free datasets.

Table 4.18 – Results of Li's segmentation and normalisation algorithms with Masek's encoding and matching routine.

Li's Segmentation and Normalisation				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	81.5%	36.5%	1.0	35.5%
WVU	53.8%	25.8%	0.8	37.6%
Average	67.7%	31.0%	0.9	36.5%
Variance	27.7%	10.4%	0.2	2.2%

The results in Table 4.18 show a significant improvement in the CASIA V1 results with better than six times the number of true matches than with Li's encoding and matching. However, there was also a small reduction in performance with the WVU Free dataset.

Overall, the average results across the two datasets are much improved from 34.5% to 67.7% true match. The separability results are still poor, though again better than before, and the very low decidability scores of 1.0 and 0.9 confirm this lack of ability to separate imposters from authentic.

The tests were repeated using just Li's segmentation code with Normalise One and Masek's encoding and matching algorithms, Table 4.19 shows the results of these tests.

Table 4.19 – Results of Li’s segmentation algorithm with Normalise One and Masek’s encoding and matching routines.

Li’s Segmentation				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	95.2%	54.1%	2.4	9.2%
WVU	95.0%	59.2%	1.9	11.7%
Average	95.1%	56.7%	2.2	10.5%
Variance	0.2%	5.1%	0.5	2.5%

Both datasets show a significant increase in recognition performance with the results Table 4.19. This indicates that Li’s segmentation is effective, and that his full flow has been let down by the normalisation, encoding and matching routines. The separability of 56.7%, decidability of 2.2 and EER of 10.5% indicates an algorithm for which, more than half of the authentic have a lower Hamming distance than the imposters. This is shown in the Hamming distribution curves for Li’s algorithm, Figure 4.15

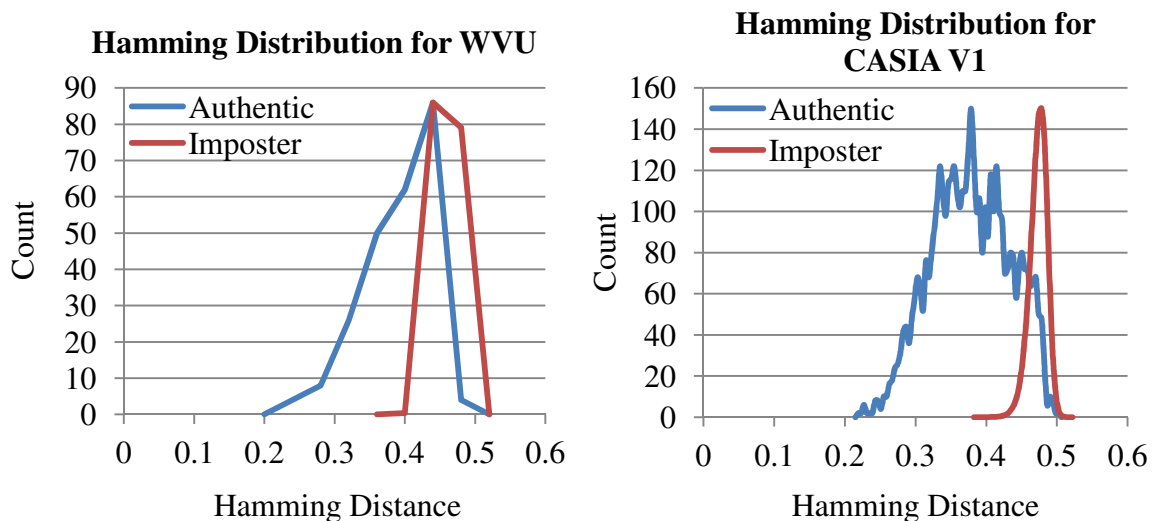


Figure 4.15 – Hamming distance distribution plots for Li’s segmentation algorithm using Normalise One and Masek’s encoding and matching routines.

Figure 4.15 shows the Hamming distribution plots, side by side for comparison, for the results obtained by Li’s segmentation algorithm. The blue authentic curves clearly begin before the red imposter curves. There is noticeable overlap, however, which is the cause of

the weak separability, decidability and EER results. These new results using Li's segmentation code with Normalise One and Masek's encoding and matching algorithms are plotted on Figure 4.16

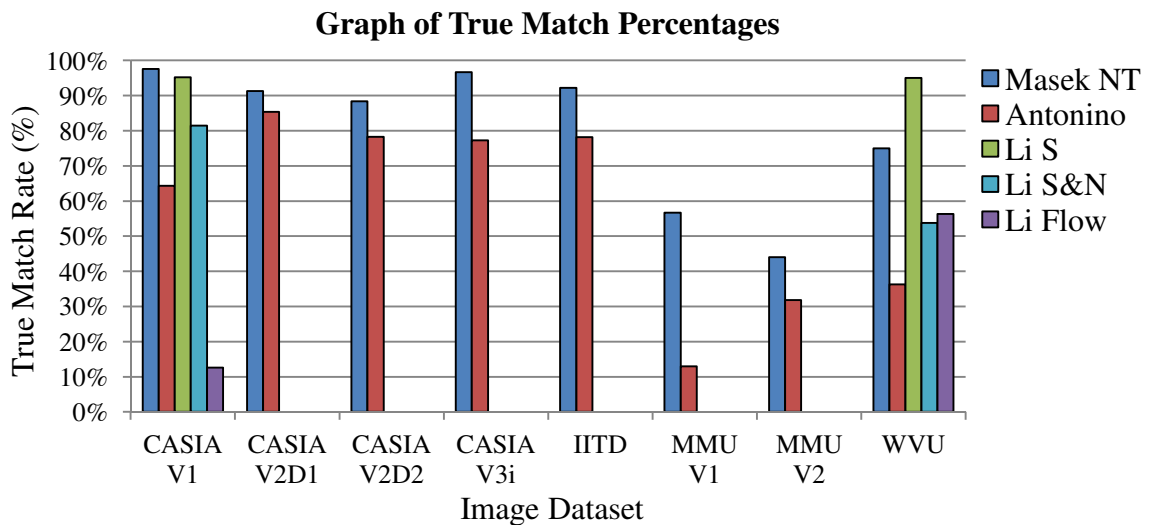


Figure 4.16 – Comparison of true match results from Masek's algorithm with no thresholding (NT) against the true match results attained by Antonino and Li's (Flow, S&N and S) algorithms.

The results shown on Figure 4.16 finally show Li's algorithm excelling with the WVU Free dataset with a 20% improvement on the results using Masek's segmentation. The CASIA V1 results are also good, though still 2.4% lower than those obtained using Masek's segmentation. The graph highlights the improvements in performance for using Li's segmentation (Li S) with Normalise One and Masek's encoding and matching as opposed to Li's normalise or full flow.

The results for Li's segmentation algorithm with Normalise One and Masek's encoding and matching algorithms are used for the remaining comparisons between algorithm results.

The problems debugging Li's algorithm led to the development of a small GUI application in MATLAB that allowed the results stored in different output databases to be compared for differences, this is discussed in Section 5.9.

The next section describes and evaluates the JIRRM iris recognition algorithm within the generic platform.

4.3.3 The JIRRM Segmentation and Normalisation Code

JIRRM [40] is an open-source iris recognition system that demonstrates a segmentation and normalisation algorithm written in Java. The JIRRM web page has this description:

“An Iris Recognition system written in Java. Acquires an image, locates the iris and produces a unique 'iris profile' for it. Will be fully supported by its own GUI but potentially linked into other applications, such as xlock.”

The code for JIRRM was last developed in 2006 and, though there was activity on the website as recently as April 2013, there has been no release since 2006. The JIRRM GUI only segments and normalises the images provided, so no actual recognition or comparison is carried out.

The JIRRM code was converted to run within the generic platform. Fortunately, the MATLAB language is very similar to the Java language that was used to write JIRRM. MATLAB even has an underlying Java Virtual Machine (JVM). This commonality simplified the translation, but significant changes were still needed.

The Java segmentation code was isolated and extracted from its application code. This was then re-factored into a segmentation routine with a separate Hough circle function. The normalisation algorithm was the same as Masek's, so was not ported to the generic platform.

Since JIRRM is focussed on the segmentation stage of the recognition process, in order for the platform to evaluate its recognition performance, the remaining three stages must be provided from elsewhere. Up to this point Masek's algorithm had provided the best performance and the greatest ability to produce a result from each of the different iris image datasets, so Normalise One was used with Masek's encoding and matching routines to complete the process.

Table 4.20 – Results from the JIRRM segmentation algorithm with Masek’s normalise, encode and matching routines

JIRRM				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	68.3%	0.7%	1.1	32.8%
CASIA V2D1	64.6%	2.9%	0.6	43.1%
CASIA V2D2	75.3%	6.3%	0.9	34.0%
CASIA V3i	59.1%	3.4%	1.0	30.4%
IITD	76.0%	-	1.4	24.4%
MMU V1	60.4%	9.8%	1.0	34.8%
MMU V2	48.5%	3.7%	0.8	36.0%
WVU	35.0%	0.8%	0.5	44.3%
Average	60.9%	3.9%	0.9	35.0%
Variance	41.0%	9.1%	0.9	19.9%

The average true match rate of 60.9% obtained when using JIRRM within the generic platform is comparable to the results obtained with Antonino’s algorithm of 58.1%. There is also significantly less variance – 41% for JIRRM compared to 72.4% for Antonino. When compared with Masek’s results of 80.2% average true match, however, JIRRM is still returning poor results.

The poor average separability result of 3.9% combined with the average decidability of 0.9 and average EER of 35% show that this algorithm struggles to separate authentic from imposters. The low variance results, especially for separability, decidability and EER, show that this algorithm is consistently ineffective.

Figure 4.17 shows the results from JIRRM against those obtained with Masek, Antonino and Li's algorithms within the generic platform.

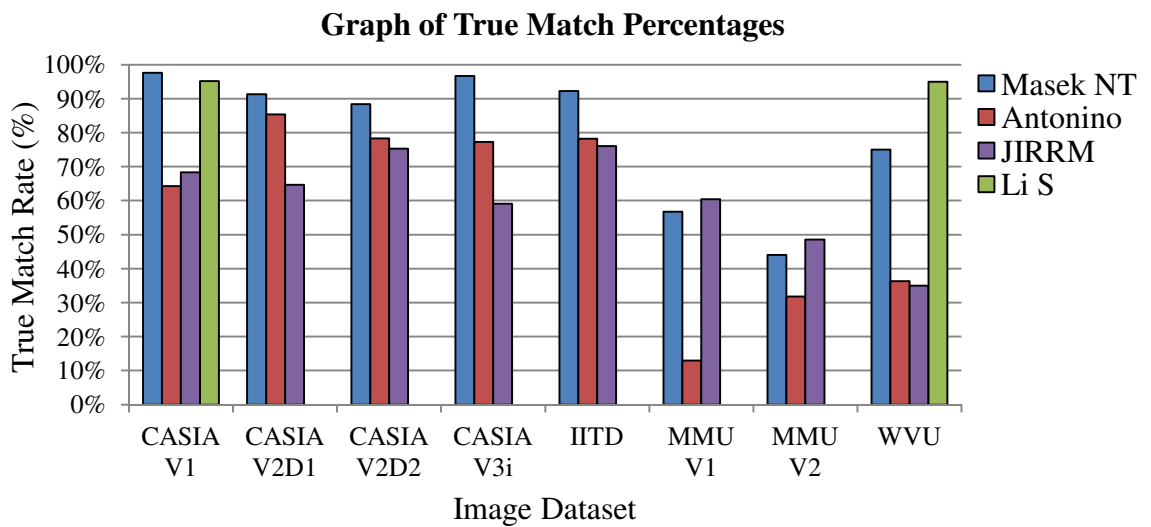


Figure 4.17 – Comparison of true match results from JIRRM against the true match results attained by Antonino, Masek and Li's algorithms.

The graph in Figure 4.17 highlights further how similar the performance of the JIRRM code is to that of Antonino. For CASIA V2D1 and V3i Antonino's algorithm clearly performs better, but JIRRM performs better for MMU V1 and V2. For the remaining datasets the results are very close with less than a 3% performance difference.

It was suspected that one reason for the relatively poor results was that JIRRM only searches for one fixed size of iris boundary and one fixed size of pupil boundary. This makes the algorithm very inflexible and unable to cope with the varying nature of iris recognition.

Since the matching for the analysis stage was carried out by Masek's algorithm, the timings for that were expected to be broadly the same as those returned for Masek's flow and this was shown to be true within the expected minor variations due to system load.

As JIRRM only looks for one circle size, it was expected that the code creation times would be much faster than Masek's. Table 4.21 shows the processing times for JIRRM using Masek's normalisation, encoding and matching algorithms within the generic platform.

Table 4.21 – Processing times for the JIRRM algorithm with the remaining test datasets.

(s) seconds, (h:mm) hours and minutes

JIRRM Processing Times			
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)	Analysis Time (h:mm)
CASIA V1	6.8	1:25	1:22
CASIA V2D1	16.9	5:38	2:52
CASIA V2D2	22.3	7:26	2:24
CASIA V3i	4.3	3:08	14:04
IITD	4.2	2:38	5:42
MMU V1	4.8	0:37	0:31
MMU V2	5.0	1:22	2:34
WVU Free	20.0	0:27	0:01
Total		22:41	35:12

Although the total code creation time in Table 4.21 was 12 hours less than Masek’s algorithm, the individual code creation times are not as definitive as was expected, as can be seen in Figure 4.18.

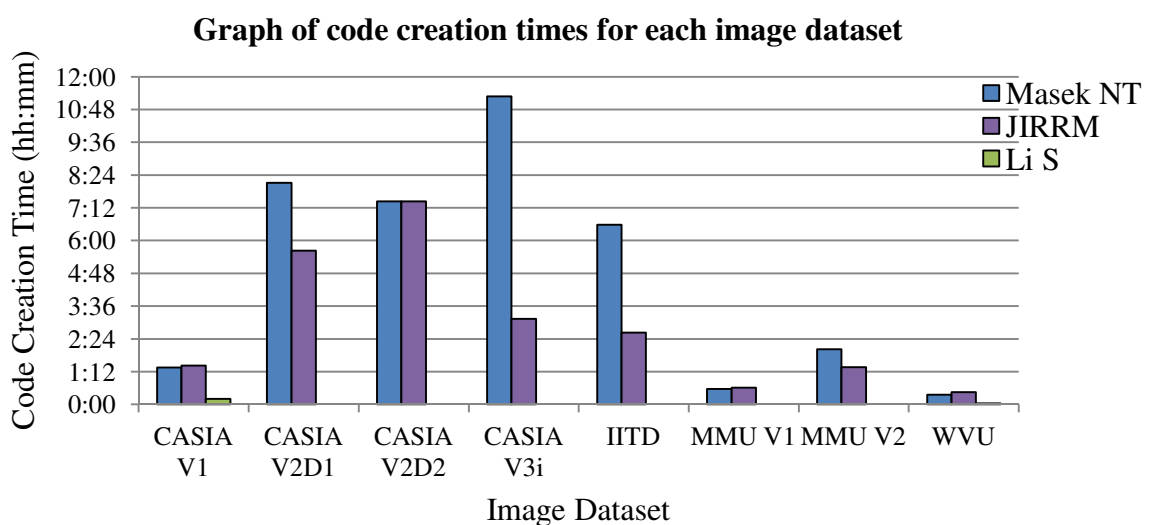


Figure 4.18 – Comparison of code creation times with Masek’s algorithm against the times attained by JIRRM and Li’s algorithms.

The results in Figure 4.18 show an interesting mix of code creation time results when compared to Masek's algorithm times (as a reference). Antonino's algorithm is not included as it uses Masek's segmentation and normalisation routine and so returns measurably similar code creation times. The CASIA V2D1, CASIA V3i, MMU V2 and IITD datasets all processed notably faster, as expected. Strangely, the CASIA V1, CASIA V2D2, MMU V1 and WVU were all about the same speed or slightly slower.

This unexpected anomaly was due to the complexity of the edge images produced. The more complex the edge image, the more circles the Hough transform had to draw and the longer the algorithm took to create the iris code.

Since the recognition results for JIRRM were poor, its chances were increased by changing the algorithm to look for more than one circle size. The range of circle sizes was set to the same as Masek's algorithm for both iris and pupil and the tests were run again.

Table 4.22 – Recognition results from the JIRRM segmentation algorithm using Masek's upper and lower pupil and iris circle sizes with Masek's normalise, encode and matching routines.

JIRRM Recognition Results - More Circles				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	81.0%	0.2%	1.5	19.54%
MMU V1	61.8%	14.7%	1.1	29.5%
MMU V2	48.5%	0.8%	0.8	36.0%
Average	63.8%	5.2%	1.1	28.4%
Variance	32.5%	14.5%	0.7	16.5%

Unfortunately, only three datasets were able to complete using the greater number of circles, as JIRRM crashed the MATLAB platform after running for several days when processing the other image datasets. This was due to a problem with the Hough routine that was, unfortunately, irresolvable without changing the algorithm completely.

Table 4.22 shows improved results across all four metrics for all three datasets using JIRRM with more circle sizes. The code creation times for this ‘improved’ algorithm were, however, also substantially increased.

Figure 4.19 compares the results from all the algorithms tested thus far with these improved results from JIRRM (JIRRM MC).

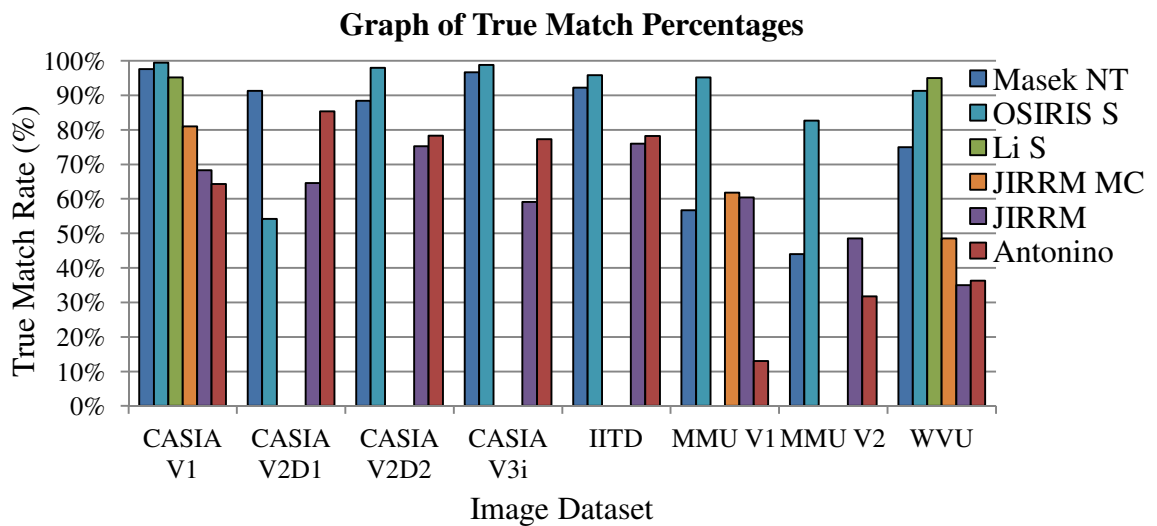


Figure 4.19 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino, JIRRM, JIRRM with more circles (JIRRM MC) and Li’s algorithms.

The true match results for JIRRM looking for multiple circles (JIRRM MC) against the original JIRRM of 12.7% and 13.5% improvements for the CASIA V1 and WVU respectively can be seen in Figure 4.19. The very small 1.4% improvement in the true match rate for the MMU V1 results is also evident. While these results are undoubtedly an improvement when compared with the standard JIRRM results, only the MMU V1 result improves upon Masek’s results and then only by a very small percentage.

Given that the algorithm failed to finish processing five image datasets, together with the relatively small gains in recognition rates, achieved, it cannot be said that the addition of more circles to the circle detection search was successful. Table 4.23 shows the code creation times for the JIRRM segmentation algorithm looking for more circles.

Table 4.23 – Processing times from the JIRRM algorithm looking for a wider range of circle sizes for both iris and pupil with Masek’s normalise, encode and matching routines.

(s) seconds, (h:mm) hours and minutes

JIRRM MC Processing Times			
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)	Analysis Time (h:mm)
CASIA V1	156.3	32:49	1:08
MMU V1	59.2	7:19	0:22
WVU Free	345.7	7:40	0:02
Total		47:48	1:32

Table 4.23 shows the code creation and analysis times for using JIRRM looking for more circle sizes with Normalise One and Masek’s encoding and matching algorithms. The code creation times were between 14 and 16 times longer than the standard JIRRM algorithm. The total for just these three datasets can be seen to be 12 hours longer than Masek’s code took to complete all eight datasets.

Overall, increasing the number of circles searched for by the Hough transform improved the recognition results, but at the cost of significant time penalties and the loss of the ability to process five out of the eight datasets. This was considered too high a price for the small gains seen and so the standard JIRRM results were used in the remaining comparisons.

The next section evaluates the OSIRIS algorithm and how it performed within the generic platform.

4.3.4 The OSIRIS Iris Recognition Code

OSIRIS [110, 39] is a C/C++ based reference design developed using the framework from the BioSecure project [143], and it is an open-source, Linux based, command line application. The OSIRIS program allows its user to script a limited variety of configuration scenarios to test the supplied iris recognition algorithm. OSIRIS was of particular interest because it was written in native C/C++ code by the research group at BioSecure. This suggested that it was the closest to a professional algorithm of all the open-source iris recognition code.

OSIRIS aims to do for C/C++ iris recognition development what Masek did for MATLAB based development, and it broadly achieves this. As the source code is available, different algorithms and modifications to the existing algorithms can be tried, though this is a more involved process than Masek's MATLAB scripts due to the C/C++ class structure used.

The OSIRIS reference design is mentioned by Tomeo-Reyes *et al.* [84], and V4.1 has been available as source code since April 2012. Its scripting file allows the input data, parameters, process steps and output data to be varied without recompilation of the source code. The key features of OSIRIS are its speed and its open-source nature. Due to the class structure used, however, it is not very flexible.

As the final open-source iris recognition codebase available it was hoped that OSIRIS could be evaluated along with the rest of the available platforms. However, the OSIRIS source code was not available until near the very end of this research, which meant there was only time to do preliminary testing.

The OSIRIS source code was compiled using Microsoft Visual Studio 2012 Express Edition and the OpenCV library version 2.4.2. The resulting executable was used to evaluate all the selected test image datasets. OSIRIS requires all the test images for a dataset to be in the same folder. The IITD dataset uses a uniform '01' to '10' naming convention for each test subject which meant that in order to coexist in a single folder all the images had to be renamed to include the appropriate subject number.

When run, the OSIRIS code outputs a text file containing the segmentation boundaries, segmented images with the noise marked on them, a normalised iris image with its mask, an iris code with its mask and the results of the matching process. Unfortunately, the

matching code only compares two adjacent images so the results are not comparable to the generic platform. However, it was possible to time the operations using the Windows command line and calculate average execution times per image, these timings are shown in Table 4.24.

Table 4.24 – Processing times for the C++ OSIRIS executable, total time in hours, minutes and seconds, average time in seconds

OSIRIS Processing Times		
Dataset	Average time / Image (s)	Total time (mm:ss)
CASIA V1	0.16	2:04
CASIA V2D1	0.19	3:51
CASIA V2D2	0.21	4:14
CASIA V3i	0.16	6:52
IITD	0.17	6:21
MMU V1	0.10	0:46
MMU V2	0.11	1:46
WVU Free	0.22	0:18
Total		26:12

Although the times in Table 4.24 are for a complete segmentation, normalisation, encoding and a single Hamming distance match per image pair, it is not unreasonable to take these figures to be an approximation of the time for the segmentation process alone. This is a reasonable approximation because experience using the generic platform has shown that the time taken by the other processes is insignificant when compared to that taken by the segmentation process.

Taking, as an example, Masek's algorithm used with the IITD dataset will help to justify this assumption. Masek's average segmentation time, measured by the generic platform, is 10.53 seconds per image. The average normalisation process takes 0.01 seconds, the average encoding process takes 0.01 seconds, and the average time for a single match operation takes 0.008 seconds. Even if those figures are subtracted from the times for

OSIRIS as they stand, 0.17 seconds for the segmentation time for the IITD images only falls to 0.15 seconds. This takes no account of the speed increase that these functions would achieve if written in C++ instead of MATLAB.

Preliminary conversion to MATLAB was completed, but only for the pupil segmentation code. Though not able to segment an image, the bulk of the main operations were implemented, allowing the process to be timed. In the generic platform, average pupil segmentation time for images from the CASIA V1 dataset was 4.81 seconds. For images from the WVU Free dataset the average execution time was 18.05 seconds.

Dividing the MATLAB execution times by the native C++ execution times returns factors of approximately 30 and 81 for the CASIA V1 and WVU Free times respectively. This gives an indication of real world performance of MATLAB iris recognition code. While not conclusive or accurate in any way, it does give the ability to assess the potential performance of an algorithm when translated to native code.

Having determined that there was insufficient time to translate the OSIRIS code from C++ to MATLAB another approach was considered. OSIRIS outputs the results of each stage of the iris recognition process to discrete files on the host computer's file system. A text file with the segmentation results, and then image files with the normalisation, segmentation mask and encoding results are all output to folders on the system's hard drive.

A short, simple script was written to import the results from the segmentation stage. This script appears in the generic platform as just another segmentation algorithm that can be selected to be used. The pseudo-code for this script is:

```
Get Image Filename
Load OSIRIS segmentation result for image of Filename
Convert pupil points to ellipse
Convert iris points to ellipse
```

The script reads the text file output by OSIRIS for the input image and outputs the parameters for the ellipses found by OSIRIS to the normalisation stage within the generic platform.

Normalise One was used with Masek's encoding and matching algorithms to complete the recognition process. Table 4.25 lists the results obtained with this configuration.

Table 4.25 – Recognition results from the OSIRIS segmentation algorithm with Masek's normalise, encode and matching routines.

OSIRIS Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	99.5%	75.5%	3.5	2.3%
CASIA V2D1	54.2%	0.9%	0.5	41.2%
CASIA V2D2	98.0%	34.7%	2.1	12.8%
CASIA V3i	98.8%	58.4%	3.8	1.5%
IITD	95.8%	0.0%	3.2	4.6%
MMU V1	95.2%	1.1%	2.7	8.4%
MMU V2	82.7%	15.1%	0.0	21.3%
WVU Free	91.3%	53.3%	1.5	16.7%
Average	89.4%	29.9%	2.2	13.6%
Variance	45.3%	75.5%	3.8	39.7%

The results shown in Table 4.25 indicate strong iris recognition true match performance. The only poor results were for the CASIA V2D1 dataset, the worst results seen from the open-source iris recognition algorithms when using this image dataset. The MMU V2 results were disappointing in comparison with the remaining six good results, but they were still significantly better than the performance of the other open-source iris recognition algorithms.

The separability results are mixed however, with the CASIA V1 results at 75.5% and the MMU V1 results at 1.1%. The average separability of 29% combined with the average decidability of 2.2 and EER of 13.6% indicate an algorithm for which less than 30% of authentications have a lower Hamming distance than the lowest imposters Hamming Distance.

Figure 4.20 directly compares the results obtained with OSIRIS against those obtained with JIRRM, Masek, Antonino, and Li’s algorithms.

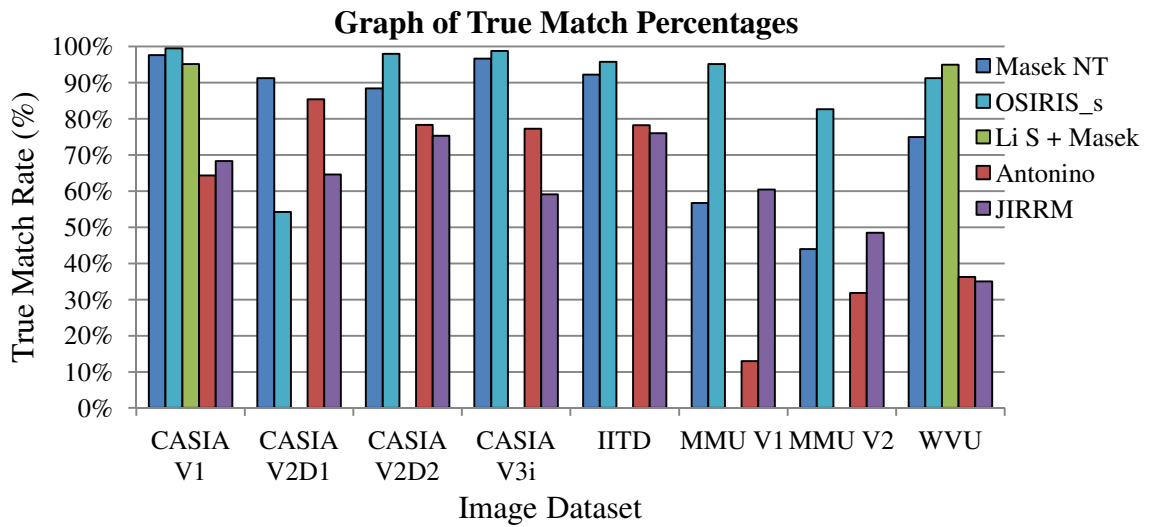


Figure 4.20 – Comparison of true match results from Masek’s algorithm with no thresholding (NT) against the true match results attained by Antonino’s encoding and matching algorithm, and JIRRM, OSIRIS and Li’s segmentation algorithms.

Figure 4.20 shows the strong performance of the OSIRIS segmentation algorithm across the datasets. The true match results are the best for all datasets except CASIA V2D1 and WVU. All of the other algorithms that were able to complete the processing of CASIA V2D1 outperformed OSIRIS by greater than 10%. For the WVU dataset, only Li’s segmentation performed better than OSIRIS and then only by 1.2%.

Clearly the segmentation process used by OSIRIS merits further consideration, but this will have to be further work.

4.3.5 Conclusions

This section discussed the process of adding, developing and evaluating pre-existing open-source iris recognition algorithms within the generic platform. The implementation and evaluation of four available iris recognition functions was discussed.

The conversion process provided a good test of the task of converting existing code to run under the generic platform and allowed further, more detailed evaluation of the remaining available iris recognition code. The comparative results obtained show how each of the five algorithms tested so far works best on particular datasets, Figure 4.21.

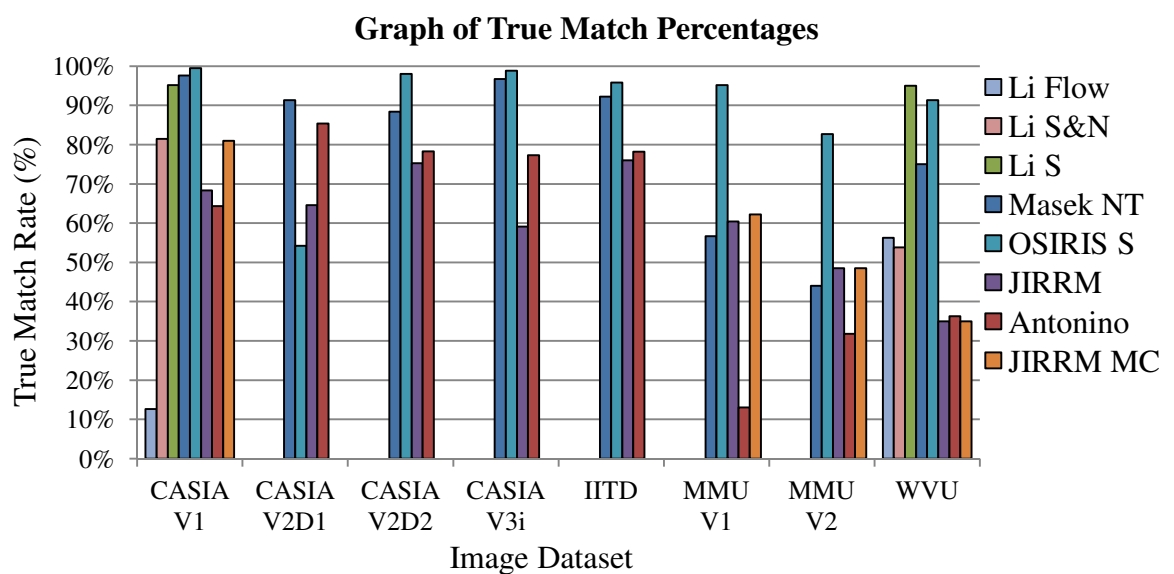


Figure 4.21 – Comparison of true match results from Masek, Antonino, JIRRM, OSIRIS and Li’s algorithms.

Figure 4.21 shows that Masek, OSIRIS and Li’s segmentation algorithms perform best on the CASIA V1. The performance of Li’s full flow on the CASIA V1 dataset is particularly weak, but performs better on the WVU Free dataset.

Li’s segmentation algorithm, coupled with Normalise One and Masek’s encoding and matching algorithms obtains the best results for the WVU Free dataset and is only slightly behind Masek’s algorithm for the CASIA V1. Li’s code was unable to complete on any of the other image datasets.

OSIRIS has the strongest performance of all the algorithms, returning the best results for six out of the eight image datasets. OSIRIS only misses the top spot for the WVU results

by 3.7%, but the CASIA V2D1 OSIRIS results are the worst for that dataset using the open-source algorithms.

Masek's code is the oldest code tested, yet it performs very well, coming a close second to OSIRIS for four of the eight datasets and having the best results for the CASIA V2D1 dataset. Masek's algorithm did well with the CASIA V3i dataset, indicating that his algorithm is not dependent on the pre-processing used in CASIA V1.

The relatively poor performance on the datasets that had high variation of illumination and capture angle suggests that his method is strongly dependant on having close-up, well illuminated, direct angle images. MMU V1, MMU V2 and WVU Free datasets do not fall into this category and this is shown clearly by Masek's weak performance with them.

The JIRRM code did not perform particularly well but did improve when the number of circle sizes that it searched for was increased (JIRRM MC). Unfortunately, this modified code was unable to complete the processing on five out of the eight image datasets.

Antonino's code never excels, always being in the middle of the pack.

The OSIRIS code conversion is a very limited test and is by no means conclusive, only the pupil segmentation code was implemented within MATLAB and it did not detect the pupil boundary as well as the C++ code, however, what this did show was just how much faster a native version could be.

The use of the OSIRIS output data to simulate full inclusion within the generic platform meant that solid, usable results for the OSIRIS segmentation algorithm were obtained. The results from this were very good indeed and established OSIRIS as the best open-source iris recognition algorithm for six out of the eight datasets being used.

Further comparisons will only include results from Masek, Li and OSIRIS, as these were the most competent.

The next section looks at the GIRIST GUI and Rosa's code and discusses why they were not added to the generic platform.

4.4 Downloadable Iris Recognition Algorithms that were not Used

There were two algorithms from Rosa [42, 43], for which the source is available, but which were not open-source nor were they integrated into the generic platform. There was also one other that was not implemented, GIRIST [44] by GruSoft which is a demonstration of GruSoft's commercial Giris SDK.

This section looks at these algorithms, starting with Rosa's 'Iris Recognition System' [43] and discusses the reasons that they were not included.

4.4.1 Iris Recognition System

'Iris Recognition System' [43] by Luigi Rosa is an optimised version of Masek's open-source code with a GUI added for simplicity. Released in April 2006 it claims a 94% speed increase over Masek's original code. The MATLAB source code is available for \$42 US, and a MATLAB p-code version with GUI compiled for MATLAB 2007 is available for evaluation.

Upon running the evaluation version within MATLAB, the program displays a GUI with eight buttons. Pressing the '*Select Image*' button opens a file-selection dialogue box where an image can be selected for loading into the GUI. Figure 4.22 shows the initial 'MENU' window next to a figure window, 'Figure 1', showing the recently loaded eye image.

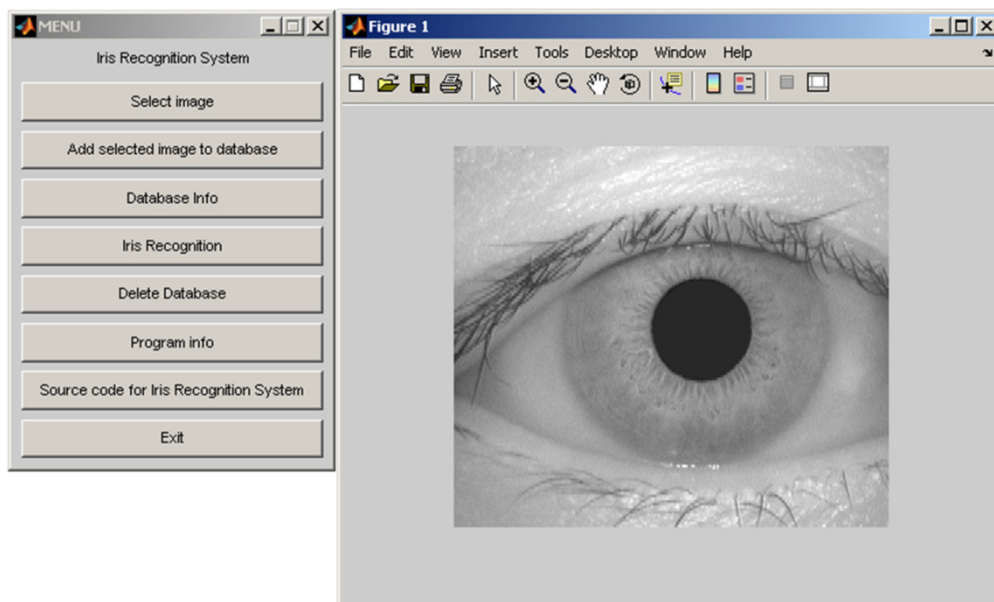


Figure 4.22 – Iris Recognition with CASIA V1 image 001_1_1.bmp loaded.

Once selected, the image can be added to the database or compared with an existing database of images. Selecting either option at this point produced an error which communication with Rosa indicated might be caused by using a newer version of MATLAB (2009a) as all Rosa's development was carried out on MATLAB version 2006a. Switching to test machine PC4 (see Table 4.2) running MATLAB 2006a allowed the code to execute correctly.

Three images from CASIA V1 were added to the database, 001_1_1.bmp, 001_1_2.bmp and 002_1_1.bmp. Four images from CASIA V1 were then compared to the database,

001_1_1.bmp, 002_1_3.bmp, 003_1_1.bmp and 004_1_1.bmp. In order, these were authentic identical, authentic same eye and two imposters.

Iris Recognition System correctly matched the two authentic but for the imposters it simply output the nearest match. Each process carried out was manually timed using a stopwatch and the whole process of adding images and then matching them was carried out ten times with the times averaged giving these results:

Average code creation time 5.9 seconds

Average matching time 7.1 seconds

The Matching time includes a full encoding cycle for the image being matched. Rosa claims that his code is 16 times faster than Masek's. Since PC4 was very much slower than PC1 used for the other testing, Masek's code was run through exactly the same tests as Rosa's on PC4:

Average code creation time 26.9 seconds

Average matching time 35.4 seconds

Again, the Matching time includes a full encoding cycle for the image being matched. To acquire these results, the generic platform was used with the same three images being added to the database and the same four images being matched against that database. These results suggest that Rosa's code is just under five times faster. This is only a limited trial but suggests that Rosa's performance claims could be ambitious.

Although Rosa responded to an initial email query, no further communication was forthcoming. The website requested payment via PayPal and then the source code would be emailed. The lack of communication with Rosa caused concern that this would not in fact be delivered.

The price of \$42, while not large, seemed poor value for money at the time for a copy of an existing open-source algorithm with a rather modest performance optimisation.

These reasons combined meant that no further investigation was carried out into 'Iris Recognition System'. With the benefit of hindsight, the decision to not pursue this code

further was incorrect. A factor of five is not a factor of 16, but it is still helpful and evaluation of Rosa's code would have allowed the manner in which the performance improvements were achieved to be understood.

4.4.2 High Performance Iris Recognition

Rosa also wrote 'High performance Iris Recognition' [42]. The website states that the algorithm uses the zero crossings of a Discrete Cosine Transform (DCT) as a means of feature extraction. This replaces the wavelet transforms used by Masek [31] and Daugman [76] in the encoding stage of the iris recognition process.

Unfortunately, Rosa has not released any literature on how exactly his algorithm uses the DCT. However, using DCT has been proposed elsewhere in the literature for the same function [144, 145, 146, 14] so the operation of Rosa's algorithm can be postulated.

The DCT is mathematically similar to the discrete Fourier transform (DFT) and can even be implemented using a DFT. Ahmed et al. introduced the linear DCT transformation in order to reduce redundant information in images [147]. The DCT generates real coefficients from a sequence of cosine functions, thus avoiding the complex numbers generated by Fourier transforms.

Applications of the DCT include lossy compression schemes such as JPEG, as well as image classification and recognition [146].

Monro *et al.* [144] were the first to propose using the phase information in the zero crossings of one-dimensional DCTs for iris recognition. Equation 4.2 was used to calculate the DCT coefficients, $C(u)$.

$$C(u) = \frac{2}{N} \sum_{j=0}^{N-1} f(j) \cos\left(\frac{2j+1}{2N} \pi u\right), 0 \leq u \leq N-1 \quad 4.2$$

where $f(j)$ is the input signal, one line of the iris image in this application and N is the length of the signal. The difference of adjacent DCT vectors is calculated allowing a binary feature vector to be formed from the resultant zero crossings.

In the literature this proposal from Monro et al is the closest to Rosa's description of how 'High performance Iris Recognition' works. Rosa's code was released in November 2008, with a claimed recognition rate of 98.84%, compared to Masek's 97.92%, when using the CASIA V1 dataset. The MATLAB source code is available for \$490 US, and a MATLAB p-code version with GUI is available for evaluation.

Upon running the evaluation version within MATLAB, the program displays the GUI in Figure 4.23.

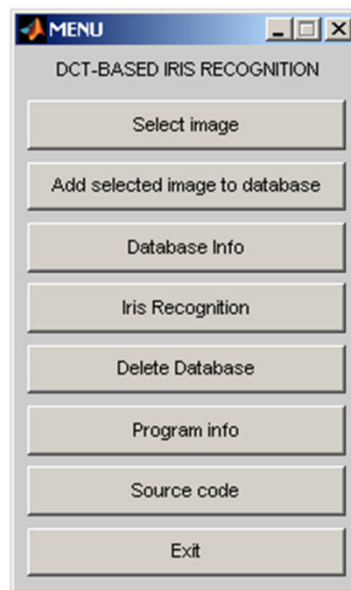


Figure 4.23 – High Performance Iris Recognition GUI

As with Rosa's Iris Recognition System, this required using MATLAB 2006a. Three bitmap images from CASIA V1 were added to the database, 001_1_1, 001_1_2 and 002_1_1. Four images from CASIA V1 were then compared to the database, 001_1_1, 002_1_3, 003_1_1 and 004_1_1. In order, these were authentic identical, authentic same eye and two imposters.

High Performance Iris Recognition correctly matched the two authentic but for the imposters it simply output the nearest match. Each process carried out was manually timed using a stop watch and the whole process of adding images and matching them was carried out ten times and the times averaged giving these results:

Encoding time	7.3 seconds
Matching time	18.4 seconds

This algorithm would have been interesting to test within the generic platform and had it been \$42 this would have been done. However, at the time, the project could not fund this amount, and a personal purchase was unaffordable. Therefore, regrettably, this algorithm could not be pursued any further

4.4.3 GIRIST

GIRIST (Grus IRIS Tool) is a commercial application from GruSoft [44, 111] that is a GUI front end demonstrating the commercial Grus SDK. While it is freely available, it is not free. The GruSoft website claims strong capabilities and high recognition rates for the tool. GIRIST allows the user to evaluate these claims with their own image datasets.

The GIRIST GUI, shown in Figure 4.24, allows the user to evaluate GruSoft's iris recognition algorithm. The GUI functions are described in the quick start guide [111] provided with the GIRIST evaluation.

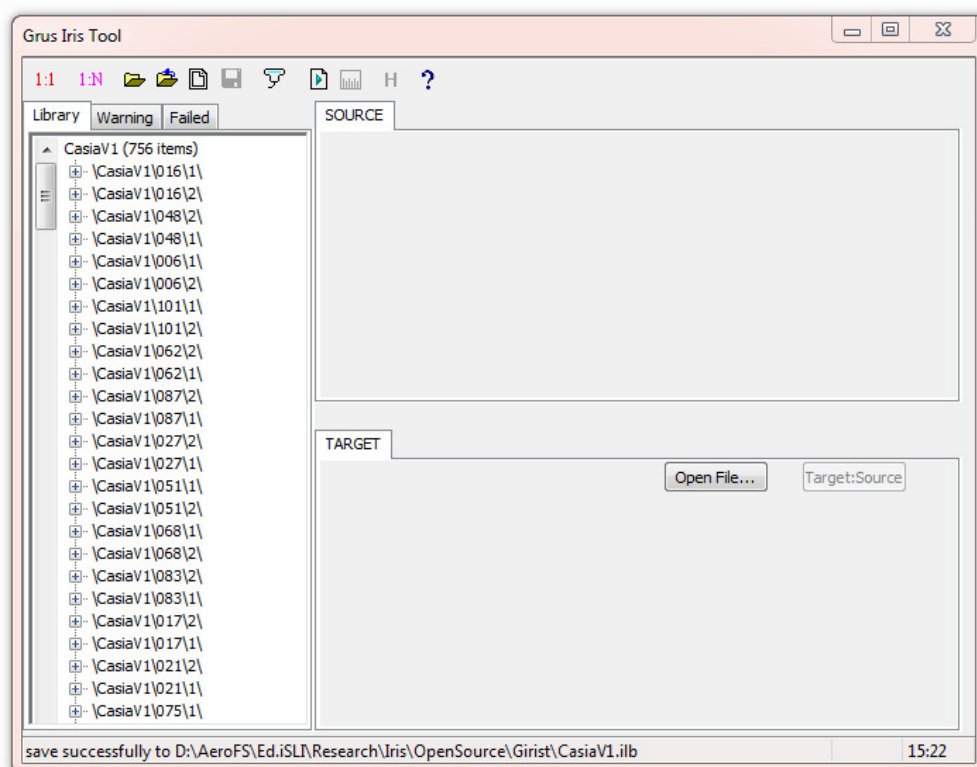


Figure 4.24 – GIRIST GUI with a library of CASIA V1 loaded

Figure 4.24 shows the CASIA V1 image dataset loaded in the 'library' tab on the left of the image, so each image has had its iris code and mask created and can be compared to any input image. A library is created for large numbers of comparisons, but individual images can also be loaded for testing.

Creating the CASIA V1 library took 1 minute 14 seconds to segment, normalise, encode each image as well as verifying the completed library on test PC1 (Table 4.2). This equates to 98ms per image or 10 images or frames per second.

This was then repeated with the CASIA V3i dataset, which took 5 minutes 5 seconds to create the library, equating to 116ms per image. GIRIST was provided with several CASIA V1 images to compare against the CASIA V3i database.

All images tested were found faster than could be measured using a stopwatch and visual inspection showed them to be the same eye image. Figure 4.25 shows the results of searching the CASIA V3i dataset using image 009_1_2 from the CASIA V1 dataset.

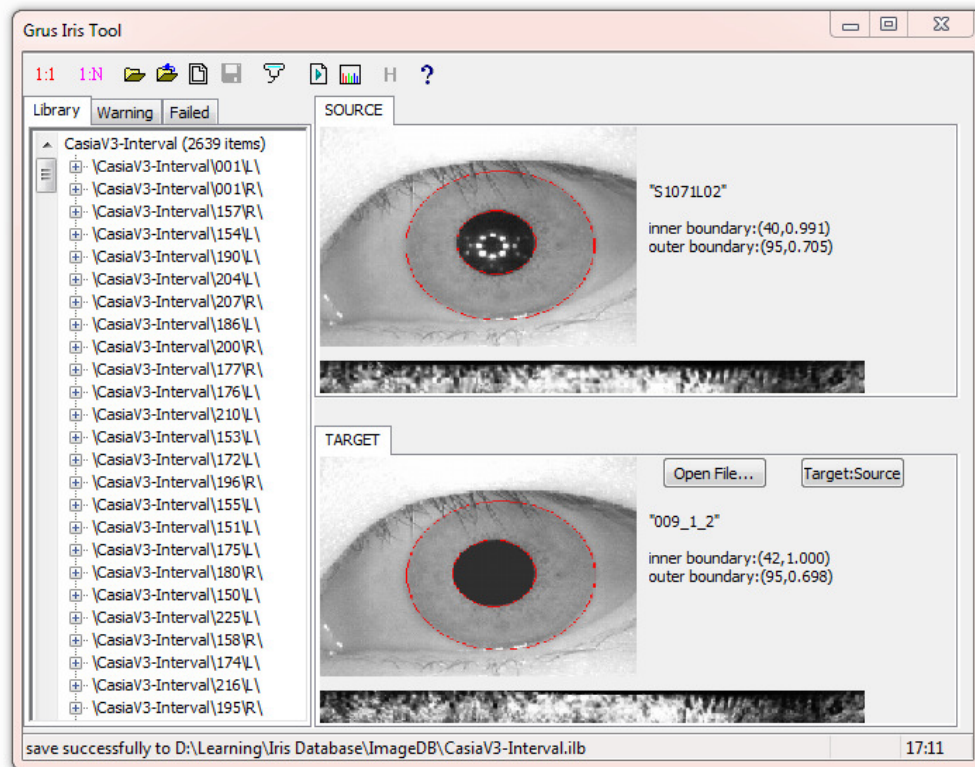


Figure 4.25 – GIRIST 1:N search result of CASIA V3 Interval database for an eye from CASIA V1 showing images and segmentation boundaries

Image 009_1_2 from the CASIA V1 dataset can clearly be seen to be very similar to S1071L02 from the CASIA V3i dataset in Figure 4.25. The obvious difference is the pupil area where the extent of the CASIA V1 pupil modification can be seen. As a demonstration of what the GIRIS SDK can do in terms of performance, GIRIST is an effective tool. Unfortunately, there is no way to evaluate different algorithms using this GUI, nor is it possible to integrate it into the generic platform.

The next section investigates the implementation of three segmentation methods from the literature within the generic platform.

4.5 Testing Segmentation Algorithms from the Literature Using the Generic Platform

Once the open-source iris recognition algorithms had been partially or completely integrated into the generic platform, three segmentation methods were selected from the literature and the effectiveness of the generic platform in comparing them to the open-source algorithms was assessed.

At the time the research was carried out, the papers by Lin *et al.* [30] and Lu [60] were the most recent literature that still used the CASIA V1 dataset. They both also appeared to rely on the histogram anomaly caused by the artificial modification of the pupils in the CASIA V1 dataset, but offered different methods of utilising this. It was of interest to see if this very fast and simple histogram methodology could be used on other image datasets.

Lu's paper was also of interest because it was the only use of the circle bisector method of finding circles in the literature, a technique that was arrived at independently in this research.

The paper from Shamsi *et al.* was selected as it used Daugman's integro-differential operator and a novel pupil-finding algorithm which gave good results on the difficult UBIRIS datasets.

The process of adding these papers to the generic platform, and the results obtained are the subject of this section.

4.5.1 Lin *et al.*'s Segmentation Algorithm

The work by Lin *et al.* [30] was addressed first because it was the most recent literature to use histogram values in detecting the pupil. None of the available open-source algorithms used this method, so it provided useful information on this type of segmentation as well as providing a suitable test for the platform. The paper by Lin *et al.* was also recent at the time this research was carried out and it used CASIA V1, leading to a fair comparison with Masek's method.

To find the pupil Lin *et al.* find the highest peak in the histogram below intensity 100 and assumes that is the pupil. Histogram thresholding is discussed in more detail in Appendix Appendix A.

To find the iris, the method from Lin *et al.* copies the pixels along two radial sectors from the source image, starting at horizontal with the pupil centre down to 20° below the pupil centre, as shown Figure 4.26.

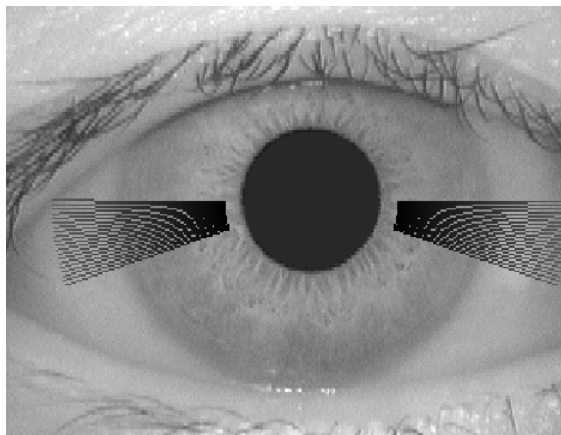


Figure 4.26 – Sectors used by Lin *et al.* for determining the Iris location marked in black on image 001_1_1 from CASIA V1 dataset

Each vertical column of the sectors shown in Figure 4.26 is summed, and the differences between adjacent column sums are calculated. The highest difference value in each sector indicates the highest rate of change and is selected as the iris boundary.

The upper eyelid is assumed by Lin *et al.*'s method to be a horizontal line placed at the centre of the pupil, so the iris above this centreline is ignored and masked off. This assumption is crude and significantly reduces the amount of useable iris, but consistently

removes the top eyelid and eyelash noise for many datasets. The lower eyelid is assumed to be below the bottom of the iris, and so is ignored. This assumption is weak as even the CASIA V1 dataset has images with occlusion from the bottom eyelid and eyelashes. It does not remove the ability to match irises, but the additional noise from these features will increase the Hamming distances calculated and thus reduce the effectiveness of the algorithm.

The next section looks at a method from Lu that uses the results of histogram thresholding in a different manner.

4.5.2 Lu's Segmentation Algorithm

The circle bisector equations were considered as a method of discerning circle parameters from edge images, so a search for prior art was started. Lu's paper [60] was the only literature found that used the circle bisector for iris recognition.

Lu presents a method that uses the same histogram thresholding idea as Lin *et al.* for finding the iris-pupil boundary. However, instead of using the extents of the thresholded area to calculate the circle, the area boundary points are determined by applying the circle bisector to a range of randomly selected points from the thresholded pixels. The circle bisector method is described in more detail in Appendix Appendix A.

The circle bisector method suffers from quantisation noise when used within a digital environment, which is almost certainly the reason Lu uses random numbers to select points. By using random numbers, the points will likely not be adjacent and sufficient spread of circles will be found such that using statistics to find the most common circle should give a good approximation to the correct one.

Lu finds the iris-sclera boundary by intensity-stretching the image such that the iris becomes much darker and the boundary more pronounced. Canny edge detection is used to detect edges within the image and the outer boundary of the iris is detected by searching outwards from the pupil for the iris-sclera edge line.

4.5.3 The Shamsi-Lin Segmentation Algorithm

The paper from Shamsi *et al.*[62] was selected because of its use of the average shrinking square method for finding the pupil centre. This does not use edge detection to find the circle centre but instead uses repeated image scaling and Gaussian smoothing. Shamsi then uses a version of the integro-differential operator, used by Daugman, to find the circle perimeter. Daugman was the first to use the integro-differential operator for iris recognition and has used it very successfully, so Lu's method was a potentially robust method.

Shamsi uses manual intervention to provide an estimate of the radius of the iris for his method before applying the integro-differential operator. Unfortunately, manual intervention is not part of the ethos of the generic platform, so another option was selected. When evaluating the method by Lin *et al.*, manual inspection suggested that their iris detection seemed to find the iris boundary whenever the pupil was correctly detected.

Lin *et al.*'s pupil detection was less successful so it was interesting to explore how effective their iris boundary detection would be with a different pupil detection routine. Thus, an algorithm was created using the pupil boundary detection from Shamsi *et al.* followed by the iris-sclera boundary detection from Lin *et al.*. Figure 4.27(a) shows image 019_1_1 from the CASIA V1 dataset correctly segmented by this method. Figure 4.27(b) shows the result of applying the average shrinking squares method to the same image.



Figure 4.27 – Iris 019_1_1 from CASIA V1 (a) segmented by Shamsi-Lin method (b) average shrinking squares applied to image 019_1_1 to find the pupil centre

The segmentation boundaries on Figure 4.27a can be seen to be close to the actual boundaries, though not perfectly on them, especially at the top of the pupil. The image in Figure 4.27b shows the result of repeatedly applying the average shrinking square method to image Figure 4.27a. The shape of the eye is still visible and the central black area for the pupil is very clear.

4.5.4 Results

Lin *et al.* and Lu's algorithms both obtained average true match rates of less than 55%, but the Shamsi-Lin algorithm only obtained 37.4%. These results were unexpected so the iris images with the calculated segmentation boundaries overlaid were inspected manually. Manual evaluation of the segmentation boundaries generated by Lin *et al.*'s method for the significantly modified CASIA V1 dataset showed that the algorithm, as described in the literature [30], often incorrectly identified one or both boundaries completely, as shown in Figure 4.28.

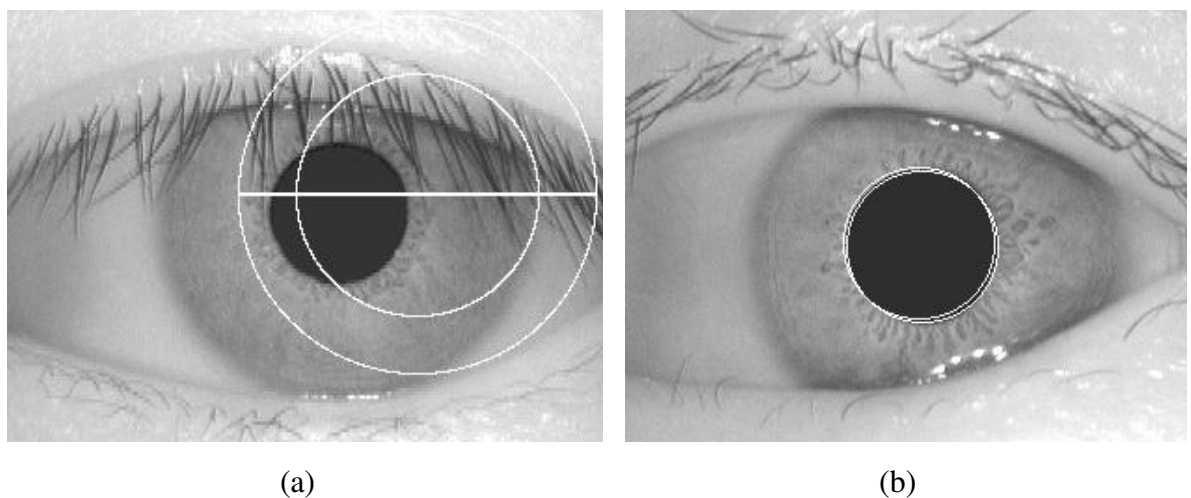


Figure 4.28 –An example of an incorrectly segmented CASIA V1 image (a) and an example of a segmented image with one incorrect boundary (b)

Manual evaluation of iris image segmentation can be subjective, especially when qualitatively comparing the results of effective algorithms, but there can be no uncertainty when the segmentation boundaries are as incorrect as those shown in Figure 4.28. Of the 756 images in the CASIA V1 dataset processed by Lin *et al.*'s algorithm, 20% were

completely wrong, as in Figure 4.28a, 56% of the images had one incorrect segmentation boundary as in Figure 4.28b and only 20% showed both boundaries correctly identified.

Lu's algorithm correctly segmented 24% of the images, but the Shamsi-Lin algorithm only managed to correctly segment 9.5% of the images within the CASIA V1 dataset. For the remaining seven datasets there were less than 5% correctly segmented images achieved by any of the algorithms.

The errors suggested that the algorithms were being confused by dark areas such as shadows and eyelashes within the images that were being detected as part of the pupil. All three algorithms struggled when darker elements were present in the corners and at the edges of the images, as they were confusing the pupil location methods. Therefore, some corrective measures were tested to see if this situation could be improved

For all three algorithms, gamma correction and intensity stretching were applied during the initial pre-processing stage before pupil detection. The aim was to separate the darker areas from the pupil intensity.

Unfortunately, this additional processing adjusted the histogram such that the peak intensity below 100 was less defined and the average true match results with this addition were lower for all three algorithms.

Instead of trying to fix the dark areas using intensity adjustments, it was considered that they could just be masked instead. The segmentation algorithms were then modified so that they added a border to each image with an elliptical 'hole' in the centre where the eye would still be visible, as in Figure 4.29.

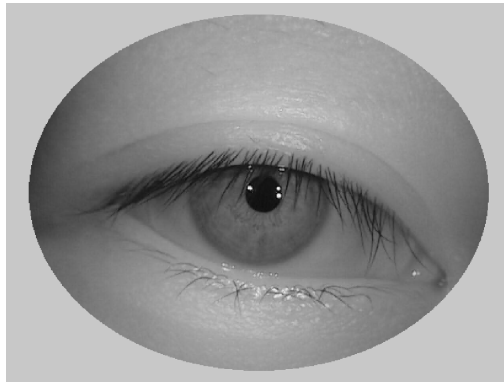


Figure 4.29 – Image 0000_012 from the CASIA V2D1 image dataset with an elliptical border added to mask darker areas from interfering with the pupil detection

The elliptical border in Figure 4.29 masks out the dark areas around the corners and sides, but still leaves the eye in the image intact and not obscured in any way. Using this elliptical masking, average true match results for all three algorithms increased. Shamsi-Lin only increased by 1%, but Lin *et al.*'s algorithm result increased by 4.5% and Lu's by 11.5%.

However, manual evaluation of the segmented images showed why the results had improved – the errors had simply become more consistent. The segmentation was being carried out wrongly, but consistently, so that the same incorrect parts of the eye image are captured each time, and these areas have a consistent (wrong) content, so matching is successful. More information on this phenomenon is given in Appendix Appendix B.

The only way of detecting this problem is to manually look at the segmented images. Unfortunately, manual evaluation of segmentation boundaries is very, boring, time-consuming, subjective and error prone.

Though it may seem pointless to compare the speeds of algorithms that do not work, there may be sections of each algorithm that could be investigated further, so a quick comparison may be of some value.

Lin *et al.*'s algorithm was very fast taking just 14 minutes to perform the code creation for all 9,560 images. In fact its speed was comparable to the OSIRIS algorithm running natively within the OS. Lu's algorithm took over 8 seconds per image, over 17 hours in total, to create the iris codes for the three image datasets it completed, so significantly longer than Lin *et al.*'s 14 minutes. The code creation times were also quite varied.

Lu's algorithm took 2.5 times longer to create all the iris codes for the CASIA V1 dataset than it took Masek's algorithm. Lu's method created the CASIA V3i dataset codes in half the time taken by Masek's algorithm, but took 20% longer to create the codes for the IITD dataset. The reason for this variation in code creation times is the variation in the number of pixels identified in the thresholding operation.

As with the method from Lin *et al.*, the Shamsi-Lin method was faster than the open-source algorithms within the MATLAB environment. All average code creation times were less than 0.5 seconds and the total code creation time for all datasets was 53 minutes. However, this performance comes at an intolerable reduction in recognition ability.

4.5.5 Conclusions

This chapter looked at the implementation and evaluation of the iris segmentation methods from three published papers.

Lin *et al.* and Lu both use a histogram thresholding method for the pupil boundary detection. Unfortunately, this does not work because histogram thresholding finds too much noise and there is no way to discriminate between noise and signal.

Even though Lin *et al.*'s pupil detection appeared to be very flawed, the iris boundary detection did appear to work. Manual evaluation of the segmented CASIA V1 images showed that, when given a correct pupil location and boundary, the iris-sclera boundary detection appeared to be reasonably successful, as shown in Figure 4.30.

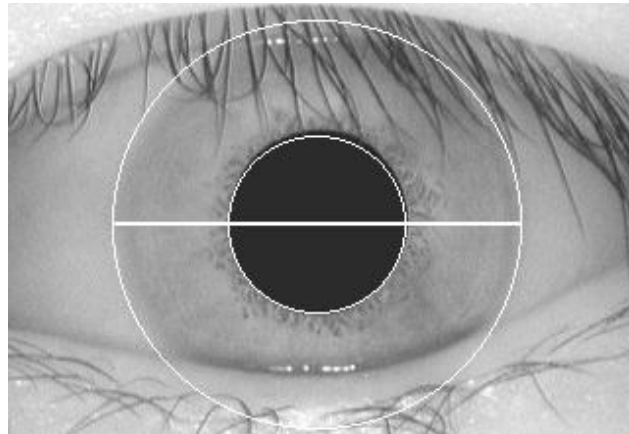


Figure 4.30 – CASIA V1 image segmented by Lin *et al.*'s method

The segmented CASIA V1 image in Figure 4.30 shows a pupil boundary with only a small amount of pupil within the iris area and an iris boundary that is slightly outside the visible iris-sclera boundary. This is a good, but not perfect segmentation result. To better understand its effectiveness this iris-sclera detection was evaluated further in Section 4.5.3 by combining it with Shamsi *et al.*'s average-shrinking-square pupil detection.

Manual investigation of the segmentation results from the Shamsi-Lin algorithm showed that the centre points found by the average shrinking square method were typically within the pupil, especially once masking was added. However, the integro-differential operation and the resulting circle radius as described in the literature did not effectively find the

correct circle radius. Without a good pupil centre and radius, the iris-sclera detection could not hope to succeed.

Lu also uses a histogram thresholding method for the pupil boundary detection, but uses randomly selected points from the pixels identified by the threshold with the circle bisector method and standard deviation to find the most popular circle to be used as the pupil circle.

The thresholding in Lu's method did not work, but the circle bisector looked interesting and is discussed further in Section 5.3. Lu uses intensity stretching and a Hough transform to find the iris boundary. This iris boundary detection was less successful than the difference of sum-of-columns from Lin *et al.*, so was not considered for further evaluation.

Although these algorithms were not effective at segmenting the iris within images, they did show how the results from the full iris recognition process cannot be trusted as the only metric for a successful algorithm, and that there is need for a reality check in the form of manual or automatic segmentation evaluation.

For this reason, towards the end of this research, an automatic segmentation evaluation method was developed, in order to help automate the process and provide additional insight into an algorithms performance. This is discussed further in Section 5.4.

The next section discusses the process of adding and supporting functions to the generic platform.

4.6 Adding Support Functions

While the ability to switch segmentation algorithms is useful, being able to, for example, change the edge detection or Hough matrix used by a function allows for much more flexibility in the experimentation and evaluation of a given recognition method.

Both the segmentation and normalisation stages have supporting functions that can be typically swapped for different methods. Segmentation often has edge detection and either circular or linear feature detection. Normalisation often has some form of pre or post processing to mask out invalid pixels, and this produces an image that can be known as the segmentation mask.

Within the generic platform, this is handled using selection list boxes. The segmentation panel, which is shown in Figure 4.31, has four list-boxes, the main segmentation list (1), the edge detection list (2), the circular boundary detection list (3) and the linear boundary detection list (4).

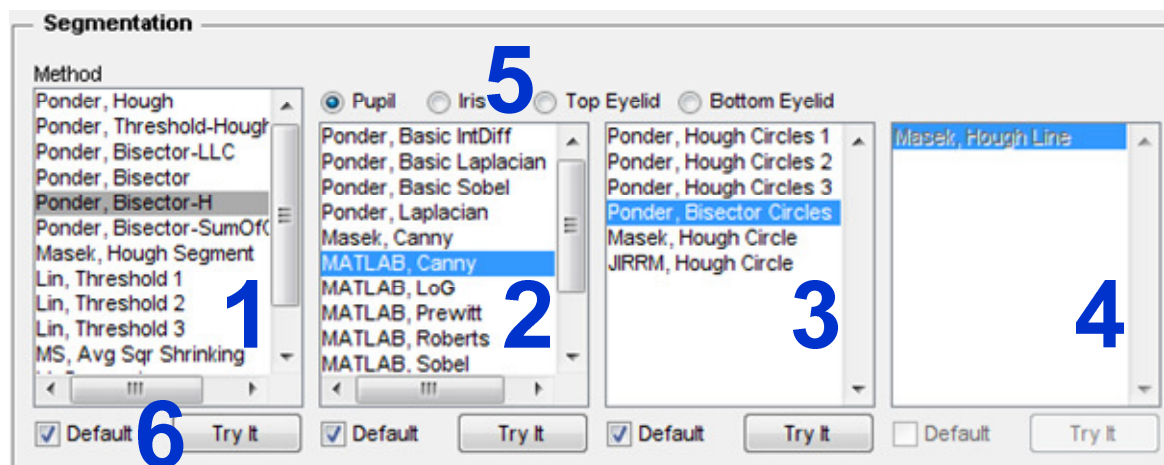


Figure 4.31 – Generic platform segmentation panel

If supported by the selected segmentation algorithm, the edge, circular boundary and / or linear boundary detection can be selected for each of the four iris boundaries of pupil, iris and top and bottom eyelids.

This is achieved by selecting the appropriate option button in Figure 4.31 (5) and then selecting the required methods for that boundary. The list boxes that are not selectable for the chosen algorithm are disabled and displayed greyed out (4). Each selectable function

can be tried using the try it button below the appropriate list (6). Setting the default check boxes (6) to ticked will force the platform to use the default options and functions.

Figure 4.32 shows the normalisation pane within the generic platform. The normalisation method (1) can be selected as can the segmentation mask (2). Again, ‘Try It’ buttons and ‘Default’ check boxes are provided (3) allowing functions to be tried and forced to default.

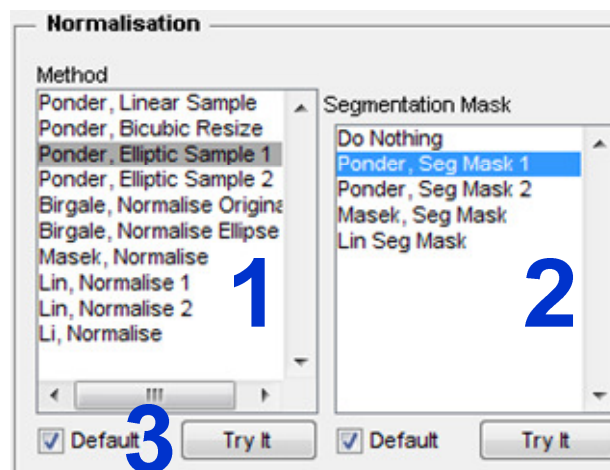


Figure 4.32 – Generic platform normalisation panel

Very little extra code is required to allow an algorithm to support swapping these lower level functions. The module informs the generic platform that function swapping is supported during the module loading process.

Including the ability to use alternative segmentation masks is provided by the generic platforms ‘ModuleRun’ command, which is a single instruction that replaces any existing segmentation mask code.

Masek’s Canny edge detection, Hough line and circle detection and segmentation mask routines were the first to be added to the generic platform. The following sub-sections detail the other algorithms that were added to the platform.

4.6.1 Edge Detection Routines

Masek provides a Canny edge detection routine that is based on a function by his supervisor Professor Peter Kovesei [148]. Masek's version of the routine has been modified to allow the edge detection to search specifically for horizontal or vertical edges. What impact would results from using a different Canny implementation, or even a different edge detection altogether?

Adding the ability to change edge detection routines was achieved by using ModuleRun, as described at the start of this section. Running Masek's flow using the newly separated Canny sub-function showed no change in the results obtained. Therefore, separating this function and including it within the generic platform caused no degradation of performance.

MATLAB includes built-in edge detection routines that can be accessed with the edge command. A mod_MATLAB module directory was created and a functions were created from the template file 'fnc_edge.m' for all five edge detections provided by MATLAB.

To confirm that the edge detection routines were working the image 001_1_1 from CASIA V1 was passed to each of the routines, one at a time, with Masek's pupil edge detection parameters. The results are shown in Figure 4.33.

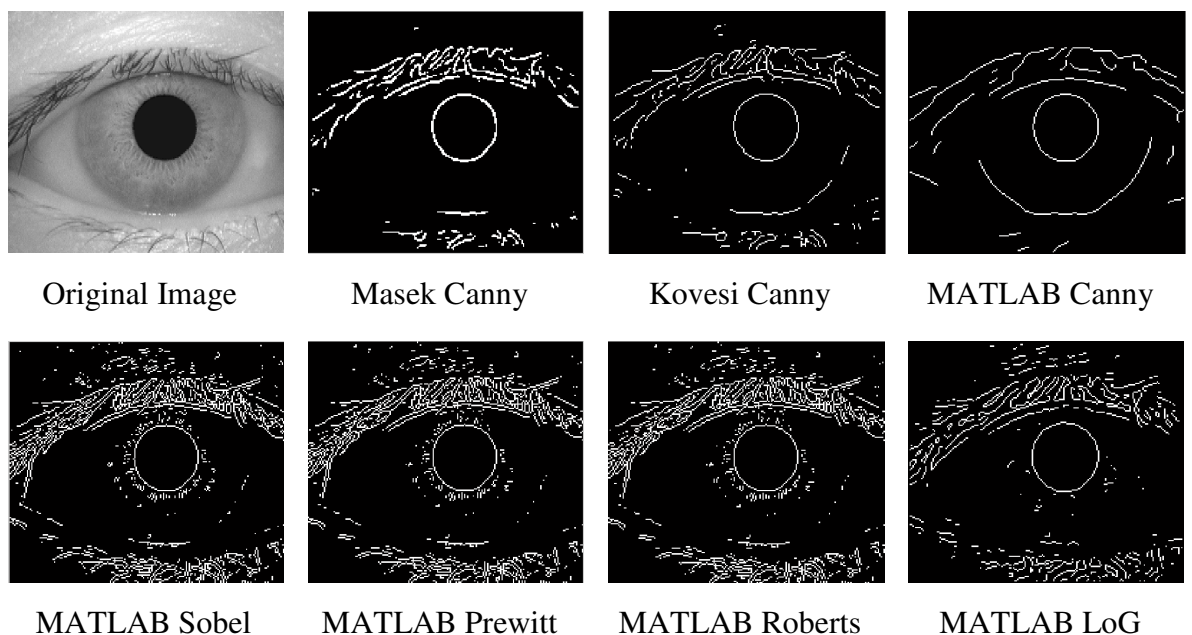


Figure 4.33 – Example edge detection images

It can be seen in Figure 4.33 that there are visible differences in most cases, though the Sobel, Prewitt and Roberts outputs look very similar. All the MATLAB edge detection routines were tried with Masek's algorithm, replacing the iris boundary detection and then again with the pupil boundary detection.

Table 4.26 shows the comparative results of using different edge detection routines for the iris edge detection in Masek's segmentation algorithm. The results obtained from running Masek's flow using Masek's Canny edge detection, with the thresholding removed, on the CASIA V1 dataset are included for comparison.

Table 4.26 – Results of Masek's algorithm using various edge detection methods for the iris boundary detection on CASIA V1 dataset

Iris Edge Detection				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
Masek Canny	97.6%	18.8%	3.2	5.4%
MATLAB Canny	95.4%	28.2%	2.6	7.1%
MATLAB LoG	94.0%	37.3%	2.5	9.2%
MATLAB Prewitt	92.5%	44.7%	2.3	11.2%
MATLAB Roberts	92.6%	41.5%	2.2	10.3%
MATLAB Sobel	91.5%	32.1%	2.2	11.1%
Variance	6.1%	25.9%	1.0	5.80

For the iris detection results in Table 4.26, replacing Masek's Canny edge detection with any of the built-in MATLAB edge detection routines led to a reduction in every recognition performance metric, except separability. This indicates poorer overall performance.

Table 4.27 shows the comparative results of using different edge detection routines for the pupil edge detection in Masek's segmentation algorithm. The results obtained from running Masek's flow using Masek's Canny edge detection, with the thresholding removed, on the CASIA V1 dataset are included for comparison.

Table 4.27 – Results of Masek's algorithm using various edge detection methods for the pupil boundary detection on CASIA V1 dataset

Pupil Edge Detection				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
Masek Canny	97.6%	18.8%	3.2	5.4%
MATLAB Canny	97.2%	36.9%	3.4	4.6%
MATLAB LoG	97.6%	40.1%	3.4	4.7%
MATLAB Prewitt	93.0%	22.1%	2.5	11.42%
MATLAB Roberts	93.0%	22.1%	2.5	11.42%
MATLAB Sobel	92.3%	22.1%	2.4	11.7%
Variance	5.3%	21.3%	1.0	7.1%

The pupil detection results in Table 4.27 show improvements in all metrics except for true match for the MATLAB Canny and LoG edge detections, but reductions when using the remaining three edge algorithms. Based purely on the results shown in Table 4.26 and Table 4.27, replacing Masek's Canny edge detection with the MATLAB LoG for the pupil boundary detection returns a significant improvement in separability, decidability and equal error rate, with no reduction in true match rate or EER Hamming threshold.

4.6.2 Hough Circle Detection Routine

Masek's code provides a Hough [41] circle detection routine which was separated out into a separate function, making it available as a circle detection choice in the generic platform GUI.

Running Masek's flow using the newly separated Hough sub-function showed no change in the results obtained. Therefore, separating this function and including it within the generic platform caused no degradation of performance.

MATLAB only provides a Hough line routine, which Masek already uses. However, JIRRM also uses the Hough circle method, this was separated out from the converted code produced earlier.

In a similar manner to the work carried out earlier with the edge detection code, Masek's algorithm was run using the JIRRM Hough code instead of Masek's Hough code for the pupil and then for the iris boundary detection. Again, this substitution of algorithms was achieved entirely through the generic platform GUI.

Table 4.28 shows the results of running Masek's algorithm within the generic platform using the CASIA V1 dataset and the JIRRM Hough sub-function for the iris and pupil boundaries in turn. The results obtained from running Masek's flow using Masek's Canny edge detection, with the thresholding removed, on the CASIA V1 dataset are included for comparison.

Table 4.28 – Results of Masek's algorithm using the JIRRM Hough for the iris and pupil boundaries in turn

Pupil Edge Detection				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
Masek Hough	97.6%	18.8%	3.2	5.4%
Iris Boundary	97.1%	34.1%	3.0	5.3%
Pupil Boundary	97.5%	33.8%	3.7	3.7%

Improvements in the EER and separability in Table 4.28 are offset by small reductions in true match rate. Using the JIRRM Hough function in place of Masek's Hough function for the pupil detection provides a large increase in separability and decidability as well as a reduction in EER for a very small reduction in true match rate.

The use of the JIRRM Hough for the iris boundary is less successful, with reductions in all metrics except separability.

4.6.3 Conclusions

This section looked at the process of adding support functions such as circle and edge detection and testing them, all within the generic platform. The first tests used different edge detection routines for the segmentation stage of Masek's iris recognition algorithm.

The results of the edge detection tests indicate that Masek's pupil edge detection results can be improved for the CASIA V1 dataset, so may be able to be improved for other datasets. Therefore, the existing algorithm has been evaluated against alternatives and a direction of further investigation has been discovered.

It must be noted that the thresholds used for each edge detection were not altered from Masek's originals in any way and so are not tuned to the particular edge detection routines used. The tests were only run on one, flawed [137] image dataset, so could be completely different on a more challenging image dataset.

The second tests used different Hough circle detection routines for the segmentation stage of Masek's iris recognition algorithm. As was the case for the edge detection tests, the results of the Hough circle detection tests indicate that Masek's pupil edge detection results can be improved for the CASIA V1 dataset. Therefore, it may be possible to achieve an improvement in recognition performance across datasets by further development of the Hough and edge detection algorithms used for the pupil boundary.

These tests also showed how even simple changes like altering the Hough transform code or changing to a different edge detection can have a measurable impact on recognition performance.

They also demonstrated how simply and quickly supporting functions can be added to the generic platform and how easily they can be swapped for alternative algorithms and used for testing.

Four areas of performance within the generic platform are discussed in the next section

4.7 Evaluating the Performance of MATLAB

MATLAB is an excellent tool for evaluating ideas and resolving complex mathematical problems, but it is not considered to be fast when used as a programming language. There were four main concerns regarding the performance of the generic platform within MATLAB, and these problems, and the testing carried out to investigate them, are the subject of this section

4.7.1 Performance of the ‘Fixed Flow’ compared against the ‘Custom Flow’

The first concern with the generic platform was that breaking iris recognition flows up into separate blocks would adversely affect recognition times. This is not specifically a MATLAB issue, as this problem could be true of any programming language. However, as MATLAB is much slower than native languages such as C/C++, it had the potential to exacerbate the problem.

Testing was carried out with Masek’s flow on the CASIA V1 and V3i datasets using both the ‘fixed flow’ and the ‘custom flow’ processes to ensure that no time penalties were incurred. The results of these tests are shown in Table 4.29 in the results section. These results show that there were no time penalties for using Masek’s code as either a ‘Fixed Flow’ or a ‘Custom Flow’.

Table 4.29 – Total times from running Masek’s algorithm as a ‘fixed flow’ and as a custom flow in MATLAB 2009a x64 on Windows 7 (PC1) against the CASIA V1 and V3i datasets

Comparative Times					
Image Dataset	CASIA V1 r1	CASIA V1 r2	CASIA V1 r3	CASIA V3i r1	CASIA V3i r2
Fixed flow (hh:mm)	02:57	02:48	02:56	25:15	24:55
Custom Flow (hh:mm)	02:54	02:50	02:53	24:59	25:07
Variation (%)	1.69%	-1.19%	1.70%	1.06%	-0.80%

The results in Table 4.29 show three runs from the CASIA V1 (r1 to r3) and two runs from CASIA v3i (r1 to r2). The times shown are the total times for code creation and analysis using PC1. The variation between the ‘Fixed Flow’ and the ‘Custom Flow’ times for the five runs is less than 2%. This figure of 2% is variation due to system load changes and neither ‘Fixed Flow’ nor ‘Custom Flow’ has consistently better times. There was no measurable change in any recognition results.

The times achieved are unusable in a real world application, which highlights the disadvantage of using MATLAB as the development platform. However, while absolute

times are unusable, comparative times can be used to evaluate if a particular method within the platform is faster than another. It must be noted however, that improvements in processing speed of 2% or less should be ignored as system load variation.

The next section looks at how these times might translate into real world performance once the algorithm is coded natively.

4.7.2 Determining the Performance Increase of Converting MATLAB code to OS Native code

The second performance concern was to estimate how fast an algorithm developed in MATLAB using the generic platform, could run when converted to a native compiled application. The results from the testing with OSIRIS, Section 4.3.4, had partially answered this, giving at least an indication of the possibilities.

There was insufficient time to carry out further testing of this process, but first indications are that converting an algorithm written in MATLAB to run natively could increase execution speed by between 30 and 80 times.

The next section evaluates the impact of the PC hardware and OS on the performance of the generic platform

4.7.3 The Performance Effects the OS and PC Hardware

The third performance concern was the effect of using different hardware and operating systems to run the generic platform. Testing was carried out to determine the performance differences across three of the machines used, PC1, PC2 and PC3. The first two are both Windows 7 PC's, while PC3 is the same hardware as PC2, but runs Linux Mint 10 instead of Windows 7. For simplicity, these machines are listed again in Table 4.30.

Table 4.30 – Test machines used to develop and evaluate the generic platform and iris recognition algorithms

Test Machines			
Name	PC1	PC2	PC3
OS	64-bit Win7	64-bit Win7	64-bit LinuxMint 10 (Gnome Ed.)
CPU	Core i7-2600K @ 3.40GHz	Core2Duo T9900 @ 3.06GHz	
GPU	nVidia GeForce GTX 550 Ti	nVidia Quadro NVS 160M	
RAM	8GB	4GB	
MATLAB Version	r2009a		

Table 4.31 shows the average processing times over five runs for the CASIA V1 image dataset using Segmentation Three and Normalise One with Masek's encoding and matching code on PC1, PC2 and PC3

Table 4.31 – Average processing times over five runs of using Segmentation Three and Normalise One with Masek's encoding and matching code on CASIA V1 dataset

CASIA V1			
Machine	Average Code Creation Time (s)	DB Code Creation Time (h:mm)	DB Analysis Time (h:mm)
PC1	0.61	0:07	1:13
PC2	0:89	0:11	1:35
PC3	0.75	0:09	1:22

PC1 with its modern, quad core i7 processor and 8GB of RAM returns the fastest results in Table 4.31. However, it is the Linux powered PC3 that is second fastest, sitting in the middle of the results from PC1 and PC2. Table 4.32 shows the results of using the CASIA V2D1 dataset.

Table 4.32 – Average processing times over five runs of using Segmentation Three and Normalise One with Masek's encoding and matching code on PC1, PC2 and PC3 using the CASIA V2D1 dataset

CASIA V2D1			
Machine	Average Code Creation Time (s)	DB Code Creation Time (h:mm)	DB Analysis Time (h:mm)
PC1	1.16	0:23	2:48
PC2	1.67	0:33	3:57
PC3	1.55	0:31	3:14

Again, PC1 returns the fastest results in Table 4.32 and PC3 is second fastest. However, for the CASIA V2D1 images the difference between PC2 and PC3 is smaller than the

difference between PC3 and PC1. Table 4.33 shows the results of using the CASIA V3i dataset

Table 4.33 – Average processing times over five runs of using Segmentation Three and Normalise One with Masek’s encoding and matching code on PC1, PC2 and PC3 using the CASIA V3i dataset

CASIA V3i			
Machine	Average Code Creation Time (s)	DB Code Creation Time (h:mm)	DB Analysis Time (h:mm)
PC1	0.58	0:26	15:04
PC2	1.17	0:51	18:39
PC3	0.75	0:32	16:43

Table 4.33 once again shows that PC1 is the fastest and PC3 is second fastest. However, for the CASIA V3i images the difference between PC2 and PC3 is greater than the difference between PC3 and PC1.

The performance tests across the three different machines showed obvious and unexpected behaviour. As expected, PC1 was consistently the fastest machine. More interesting was the consistency of the Linux machine, PC3, when compared against the two Windows 7 machines. The CASIA V1 and V3i datasets both contain images at 320×280 pixels and both return average times of 0.75 seconds per image on PC3 yet vary on the other two PC’s. While not conclusive, this suggests that more consistent timing can be achieved with the generic platform if it is run under Linux.

The generic platform running under Linux on PC3 was also consistently faster than when it was run on the PC2. These two machines are the same hardware running different operating systems. This suggests that Linux Mint 10 is a faster platform for testing than Windows 7.

The next section evaluates a method for boosting the performance of the generic platform

4.7.4 Boosting Performance of the Generic Platform Using Pre-calculation

Finally, as has already been stated, MATLAB is very effective in its role as an advanced graphic calculator, but as a general-purpose programming language, it is very slow. Matching and segmentation are typically the most time consuming stages out of the four stages of iris recognition, often taking many hours in MATLAB. This can make testing and evaluation a very lengthy process.

One option is to write algorithms in the optimal manner for the language and specific algorithm. While writing in the optimal manner will generally speed up an algorithm, if the underlying method is slow, no amount of code optimisation is going to make it run fast. One solution to this is to reduce the number of images being tested, but this skews the results. This was shown in Section 4.1, when Masek's code running under the generic platform was compared with the results in his thesis.

Another solution, when using a staged process like iris recognition, is to use pre-calculation. Pre-calculation also has its drawbacks - the pre-calculated data must be stored and, for iris recognition, each stage is dependent upon the previous, so only the start of the process can use pre-calculation. This means that if the stage being developed is segmentation, pre-calculation cannot help as every change to the code means that the pre-calculation data must be re-created.

However, if the stage being developed is normalisation, encoding or matching, having the lengthy segmentation process pre-calculated can reduce the time required for testing by several hours. This leads to a faster turn-around for results, which is always helpful so long as it does not influence accuracy in any way.

For these reasons, pre-calculation was considered a useful tool to have within the generic platform. Maintaining accuracy was resolved very quickly as, during the automated encoding process the results from each segmentation or 'fixed flow' are stored to a separate structure and at the end of the encoding phase are stored to the root directory of the result database.

By using automatically generated file names and a fixed storage location within the host filesystem for the pre-calculation files, the pre-calculation was made simple and automatic

for the user. This method was consistent and repeatable and avoided any chance that the user could select the wrong pre-calculation file for the selected algorithms.

Table 4.34 – Time for code creation using Masek’s algorithm on the CASIA V1 dataset, without and with pre-calculation on PC1

Comparative Times		
Image Dataset	CASIA V1	CASIA V2D1
Without Pre-calculation	1h 21m	8h 7m
With Pre-calculation	17s	45s

The results in Table 4.34 show that with pre-calculation the code creation stage of the iris recognition algorithm testing is significantly faster. There were no measurable differences in any other results, even the execution times output at the end of the process were correct to the original segmentation process.

Reducing the code creation time from hours down to seconds makes testing much faster, which can be a huge help for the developer. The addition of ‘Force’ check box means that the developer is in control of whether pre-calculation is used or not.

The conclusions for the evaluation of the performance of MATLAB are presented next

4.7.5 Conclusions

This section outlined the performance concerns of ‘fixed flow’ against the ‘custom flow’, the possible real world performance of algorithms developed in the generic platform, the effect of the PC hardware and operating system used on performance and a more general performance problem with MATLAB.

The execution times of Masek’s ‘fixed flow’ against Masek’s ‘custom flow’ within the generic platform highlighted the performance disadvantage of using MATLAB. The OSIRIS performance increase of between 30 to 80 times fast for running natively that is discussed in Section 4.3 allowed the comparative times output by the generic platform to be evaluated in terms of potential real world performance.

The hardware and operating system tests showed the expected performance increase for faster hardware, but the unexpected increase in performance and consistency for using Linux instead of Windows suggested that Linux might be the better option during development.

The use of pre-calculation for the segmentation stage within the generic platform really improves the ability to carry out testing within the generic platform for the development of normalisation, encoding and matching algorithms.

The overall conclusions for this chapter are discussed next.

4.8 Chapter Conclusions

This chapter discussed the evaluation of the generic platform using open-source algorithms and implementations from the literature. The inclusion of support functions was tested and the performance concerns with respect to MATLAB were discussed with some partial solutions being presented.

The ability to integrate iris recognition algorithms into the generic platform is relatively straightforward for code written in MATLAB, even the Java code of JIRRM presented no significant challenge. However, converting the C/C++ code using the OpenCV image-processing library proved to be more challenging. The biggest challenge was that MATLAB returned small but measurable differences in the results of operations on data as compared to the OpenCV equivalents. This was irresolvable without the very significant task of re-writing the OpenCV routines in MATLAB.

The ability to re-use the results output from the segmentation stage of the OSIRIS OS native executable ensured that, while execution times were not usefully comparable with the code running natively within the generic platform, the recognition results could be compared. This opens new possibilities for evaluating recognition performance between algorithms, even for those that cannot be easily ported to run within the generic platform.

Implementing algorithms in software is a difficult and time-consuming process, irrespective of which programming language is used. It is often made more difficult because algorithms presented in the literature are incomplete, as the author only chooses to present the parts that he or she feels are important.

The generic platform can simply implement and test full, or partial, algorithms, combining them with, or adding them to, other algorithms as required. These features have been used heavily in this chapter, and it is only because of the availability of the generic platform that these tests could have been completed, and conclusions drawn, within a realistic timescale.

For all three algorithms implemented from the literature, problems were seen in the algorithms ability to cope with darker image areas that were not the pupil, so they failed to segment the majority of images correctly. The results from Lin *et al.* and Lu's algorithms showed fixed intensity histogram thresholding to be a flawed concept.

Discovering the problems with the segmentation results required significant time to be spent manually evaluating the segmentation overlay images of the 9,560 images from eight datasets that were used for testing. The generic platform simplifies this evaluation process in automatic mode, as it can output each image with its segmentation boundaries overlaid. However, even though these images provide a reality check against poor algorithms producing good results with datasets that have minimal image variation, it is still a very significant mechanical task. This challenge prompted the investigation into automatic segmentation evaluation discussed in Section 5.4.

It was interesting to see the existence of “Iris Recognition System” and “High Performance Iris Recognition”, being paid for MATLAB iris recognition algorithms. However, the cost of acquiring the source code and the author’s unwillingness to engage beyond a cursory initial response meant that these were not considered further. It is accepted that in the case of “Iris Recognition System”, which was not very expensive, this was a mistake.

Looking more closely at the individual results for all of the tests performed, there is significant variation across the image datasets, with clear preferences with respect to lighting conditions, distance from the camera and angle of eye with respect to the camera. Only OSIRIS maintains true match results greater than 90% for more than half the datasets, though Masek’s algorithm does well with four of the eight datasets. This seems restrictive and it would be more useful to have an algorithm that was effective across a broader spectrum of datasets.

When compared to manually writing the code for every aspect of the process the generic platform is significantly simpler and faster. The test results from the generic platform show consistent and clear output. Each result is directly comparable to the others allowing fast evaluation of an algorithms performance. This is also true for testing different support functions without changing the rest of the flow.

There is no measureable time penalty for using the platform against not using it. In fact, when the labour intensive methodology often employed by researchers [62, 60] is considered, the generic platform increases accuracy and reduces evaluation time.

The next chapter describes and evaluates the algorithms developed as part of this work

5 Algorithms Developed as Part of this Work

While developing the generic platform [112] and researching and evaluating existing algorithms, seven algorithms for iris recognition, an automatic segmentation evaluation algorithm and a results comparison GUI were developed. This chapter focuses on that work.

The focus of development for the iris recognition algorithms presented in this chapter was increased performance, especially within the generic platform environment, and strong recognition performance across all eight datasets used for testing. It was reasoned that if an algorithm was fast within the generic platform as compared with other generic platform algorithms, then it had the potential to be very fast once converted to run natively.

The first three algorithms discussed are segmentation algorithms that build each upon the last, starting with Segment One and culminating in Segment Three. Since they are dependent, they are presented in the order that they were developed. The first of these, Segment One, is a Hough transform [41] based algorithm using Canny edge detection [45]. The novelty in this algorithm was the use of a single edge detection and a single Hough transform operation to find both iris and pupil boundaries.

The second segmentation algorithm, Segment Two, also uses the Hough transform, but only for the pupil detection. Segment Two uses Otsu thresholding [46] to help minimise the number of points in the edge image, thus reducing the Hough transform processing time. The iris boundaries are determined using a sum-of-columns method. The novelty of this algorithm is the use of Otsu thresholding to help find the pupil and the combination of Gaussian smoothing with the horizontal Sobel convolution matrix to determine the iris boundary.

The final segmentation algorithm, Segment Three, uses a novel method of selecting edge points from a Canny edge image for use with the circle bisector method to find the pupil and eyelid boundaries. Additional averaging and weighting operations are then used to improve the accuracy of the circles discovered. The iris boundary is determined using the same sum of columns method that was created for the second method, but with additional Gaussian smooth operations.

After the segmentation algorithms, an automatic segmentation evaluation algorithm is described. This uses the results from all the segmentation algorithms tested during this research to evaluate the segmentation boundaries of algorithms under test. This algorithm appears to be unique, with no other attempts to detect automatically the quality of iris segmentation boundaries found in the literature.

Section 5.5 introduces Normalise One, a normalisation algorithm created to replicate Masek's normalisation code but extend it to cope with elliptical iris boundaries and circular and elliptical eyelid boundaries.

The following section discusses the development of Normalise Two. This section investigates three modest enhancements to the normalisation process, and compares the results obtained when using them with Normalise One. It then goes on to combine these modest enhancements into one normalisation algorithm to see if it will improve the recognition accuracy.

A more flexible segmentation mask routine to replace Masek's is then discussed in Section 5.7. This routine, Segmentation Mask One (SM1), removes the thresholding present in Masek's segmentation mask and adds the ability to mask elliptical and circular eyelid boundaries.

The next section introduces Match One. This algorithm shows a method of increasing speed using a simple algorithmic improvement, which therefore has no effect on the matching performance when compared to Masek's method. It was tested using Masek's segmentation, normalisation and encoding.

Normalise One, SM1 and Match One have been shown to have no measurable effect on the recognition results when using Masek's routines, but together they provide greater flexibility of boundary shape and significantly reduced analysis time.

The penultimate section in this chapter presents a small GUI that allows the comparison of results databases. This is very useful for detecting where differences are within results, especially for 'Fixed Flow' against 'Custom Flow'. This GUI was created to help with the debugging process when integrating Li's algorithms.

The final section in this chapter compares the methods and algorithms developed for this work with those that were current in the literature at that time.

The algorithms and methods discussed in this chapter are each dependent upon the previous. For this reason, each method will be described individually, with the results obtained by it, and then the next method will be described.

As stated for Chapter 4, the separability results for the IITD dataset are consistently 0.0% for every single algorithm tested. This appears to be due to an error within the dataset having two copies of the same image labelled as being different eyes. There was insufficient time to track down the erroneous image so the result is presented but ignored in the text and any separability averages presented.

The code creation and analysis times presented in this chapter are measured by the generic platform but are dependent upon both the algorithm implementation as well as the system load. As MATLAB code executes much slower than real-time speeds real world evaluation of the performance is difficult, however, the timings are suitable for the comparison of the performance of one algorithm run within the generic platform against another algorithm also run under the same generic platform environment.

The chapter builds towards an iris recognition flow, named Flow One, that uses Segment Three, Normalise Two, Segmentation Mask One and Match One with Masek's encoding algorithm. At the end of this process all the methods will be summarised, with their results.

The next section discusses the first segmentation algorithm

5.1 Segment One

The purpose behind the Segment One algorithm was purely investigative, to develop a better understanding of the Hough circle transform and the use of edge detection.

Based on the performance of JIRRM and Masek's algorithms, it can be seen that the Hough transform is a very slow operation when executed in MATLAB. However, it is also a very effective method of finding circles in images, so an improvement in speed would be beneficial. The circular Hough transform is described in more detail in Appendix Appendix C.

Using a single circular Hough transform to find the iris and pupil boundaries on the same edge image, in theory, would be faster than searching for the two individually. With the iris and pupil boundaries on the same edge image, the strongest circle found by the Hough transform should be the iris boundary and the second strongest should be the pupil boundary. This, was the process used for Segment One, but an edge detector was needed.

The Canny edge detector was selected for the edge detection [45] as it is widely regarded to be the optimal edge detection routine for image processing applications. The Canny edge detector filters out noise by applying a smoothing filter, such as a Gaussian filter, then the Sobel operator is applied to the smoothed image to determine the gradients across the image. These results are then processed to find the local maximums, and then hysteresis is used to remove values below the threshold.

The popularity of Canny edge detection means that it is very well documented in academic papers and on the internet, so will not be covered further here. The particular Canny edge detection algorithm used was the one available through MATLAB's 'edge' command. This command takes three parameters, the high and low thresholds for the hysteresis and a sigma value for the smoothing.

In order to understand the impact on the edge image of varying the upper and lower thresholds and the sigma value used in the Canny edge detector, tests were run on ten randomly selected images from each of the eight image datasets. Each image was passed to the edge detection routine multiple times with different values for the thresholds and sigma. The thresholds were varied from 0.05 to 0.8 and sigma from 0.5 to 3.0.

The aim was to determine a common set of thresholds and sigma value that returned a clean edge image with both iris and pupil circles shown. Figure 5.1 shows two of the images that were tested with their respective edge images overlaid and shown separately.

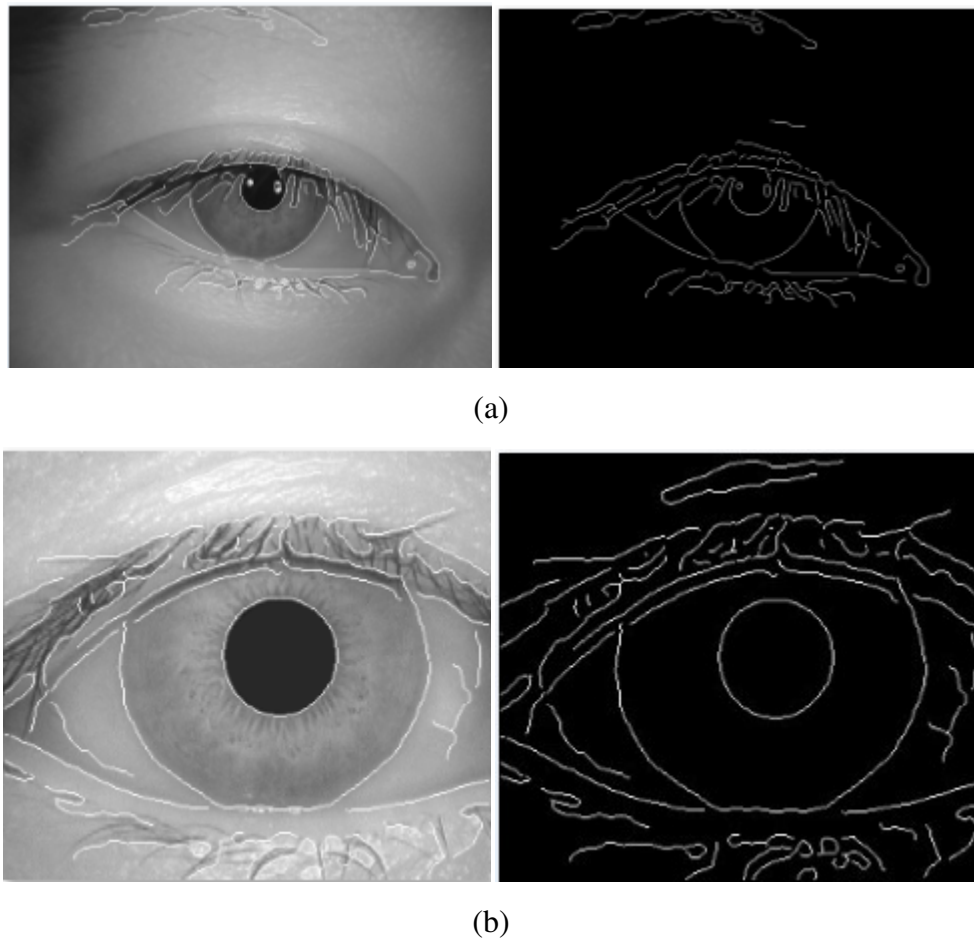


Figure 5.1 – Sample images with their respective canny edge images for low threshold of 0.08, high threshold of 0.15 and sigma 2.5. Each image has its edge image overlaid (a) CASIA V2D1 image 0002_008 and (b) CASIA V1 image 001_1_1.

The iris and pupil boundaries are clearly visible on both images in Figure 5.1. Varying the value of the thresholds altered the number of edge points, but increasing sigma up to 3.0 increased the number of linked points.

Manual inspection of all the tested images with their edge detection overlaid, showed that a low threshold of 0.08, a high of 0.15 and a sigma of 2.5 might produce images with both boundaries clearly visible across the eight image datasets used for testing. The results of using the Hough transform with Canny edge detection using these parameters are presented next.

5.1.1 Segment One Results

The recognition results are listed in Table 5.1 and show that overall, performance of Segment One was poor.

Table 5.1 – Recognition results from Segment One using Normalise One, Segmentation Mask One, Masek’s encode and Match One

Segment One Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	57.8%	0.2%	1.0	33.5%
CASIA V2D1	78.8%	6.4%	0.9	34.7%
CASIA V2D2	47.7%	4.6%	0.8	36.3%
CASIA V3i	60.4%	2.1%	1.1	31.7%
IITD	72.4%	-	1.4	25.1%
MMU V1	59.8%	2.1%	1.1	32.6%
MMU V2	21.0%	1.0%	0.5	43.6%
WVU Free	35.0%	1.7%	0.5	40.0%
Average	54.1%	2.6%	0.9	34.7%
Variance	57.8%	6.2%	0.9	18.4%

Across the image datasets, the average recognition rate was 54.1%, this is similar to the results from other open-source code that was available at the time Segment One was created. It is slightly lower than JIRRM at 58.7% but better than the Shamsi-Lin code at 37.4%.

If modifications are included, however, Masek’s code with the thresholding removed significantly outperformed Segment one with an average recognition rate across the image datasets of 80.24%. In fact only the Shamsi-Lin code with elliptical masking was worse at 38.8%. Separability of 2.6% combined with decidability of 0.9 and an EER of 34.7% show an algorithm that struggles to separate authenticics from imposters.

Table 5.2 shows the code creation times for Segment one using Normalise One, Segmentation Mask One and Masek's encode.

Table 5.2 – Code creation times of Segment One using Normalise One, Segmentation Mask One and Masek's encode

Segment One Processing Times		
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)
CASIA V1	12.5	2:38
CASIA V2D1	22.9	7:39
CASIA V2D2	18.0	5:59
CASIA V3i	11.7	8:34
IITD	11.6	7:12
MMU V1	15.0	1:52
MMU V2	22.4	6:11
WVU Free	52.0	1:09
Total		41.19

The code creation times for the CASIA datasets are faster than Masek's code achieves but the IITD, MMU and WVU Free results are all slower than Masek. Overall, Using Segment One as part of the encoding process increased the total amount of time to encode all the datasets. There was no impact on the total analysis times for using this segmentation algorithm, so they are omitted.

The reason for the poor recognition performance becomes apparent when the segmented images are examined. Figure 5.2 shows nine images from the CASIA V1 dataset that have been segmented using Segment One.

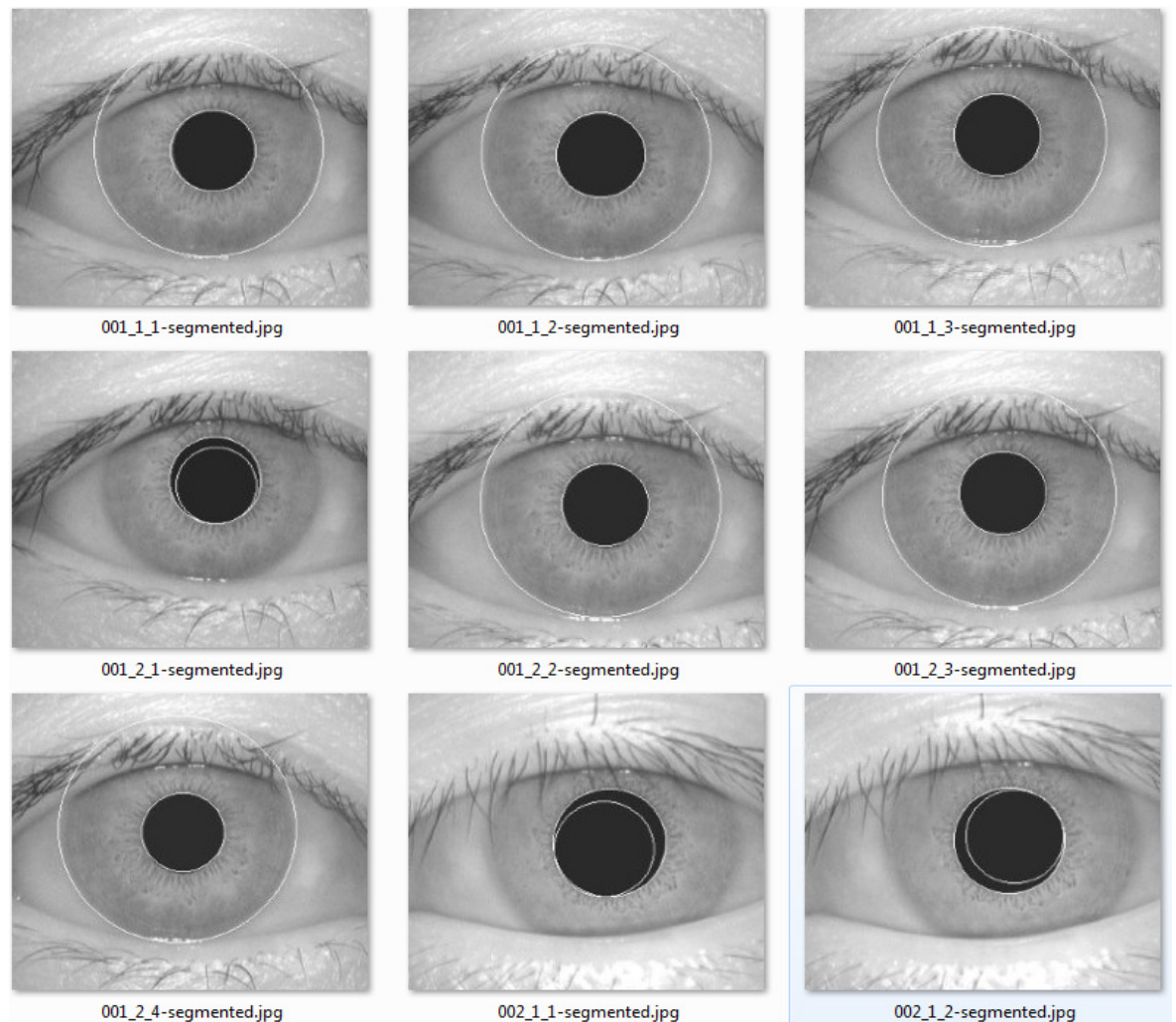


Figure 5.2 – Images from CASIA V1 segmented by Segment One

It appears that for images 001_2_1, 002_1_1 and 002_1_2 in Figure 5.2 that this algorithm has found the pupil circle first, when it is looking for the iris. Thus, when it looks for a smaller circle that might be the pupil it finds a variation of the pupil circle.

Segment One was able to complete on all the datasets and overall it was only slightly slower than Masek's code. Recognition performance, however, was consistently poor managing an average of only 55.6% true match rate across the datasets. This compared poorly both to the individual results and the average of 80.2% for Masek's code.

Overall, the Hough transform performed well, typically finding one of the circles accurately. However, it was easily confused by the presence of both circles in the edge image. The results from testing Segment One prompted the development of Segment Two, which is discussed next.

5.2 Segment Two

Manual evaluation of the Segment One results showed that the pupil was located more consistently than the iris. This suggested that it might be easier to find the pupil circle first. The Hough transform was used again, but this time its efforts were focused on finding the pupil.

The Hough transform [41] is often used to great effect [31] to search for shapes in an image, however, as seen with Segment One it tends to be quite slow. While there have been attempts to speed up the Hough transform [149] the simplest way to speed it up is to reduce the number of circles it has to look for. The encoding time performance of Segment One showed that this was necessary.

The work by Lin *et al.* and Lu used thresholding that was particularly dependant on the CASIA V1 pupil modifications, but it was of interest to determine whether a more flexible thresholding method could be effective. The Otsu thresholding method [46] was used to investigate if that would be more effective than looking for peaks in the histogram.

Otsu thresholding uses probabilities to determine a pixel intensity threshold with minimum intra-class variance between background and foreground. Intensity values below the threshold are considered to be background information, values above the threshold are considered to be 'objects'.

The MATLAB Otsu implementation also returns an effectiveness value indicating how well the input image is thresholded. Combining the returned threshold with the effectiveness returns a threshold that includes the eyelashes and pupil in the background and everything else in the foreground. Converting the source eye image to a binary image using this threshold, with background as 1's and foreground as 0's, gives an image like Figure 5.3.

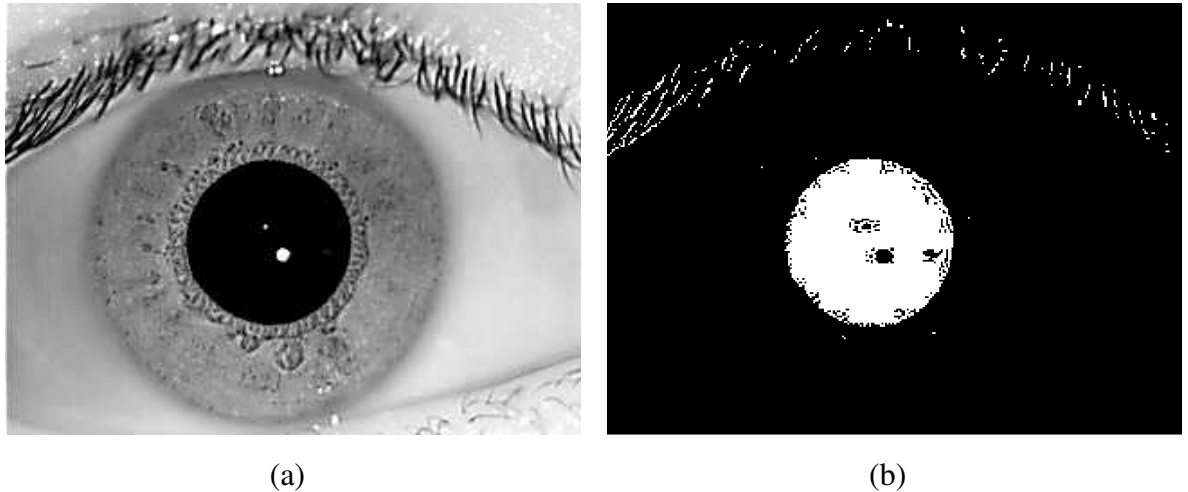


Figure 5.3 – (a) Image 001/02 from the IITD image dataset and (b) its Otsu threshold image

The image in Figure 5.3b shows a strong solid disc at the centre where the pupil from Figure 5.3a is located, and smaller features across the top that are the eyelashes. Using this disc with the Hough transform would, unfortunately, return a great many strong circle candidates, and would be very slow, so some method of reducing the number of points was needed.

Canny edge detection [45] is remarkably good at finding the edges in an image. Unfortunately, the pupil is not the only edge in an eye image. This result can be adjusted by changing the thresholds, however, earlier work looking at thresholds showed that it was difficult to get a consistent edge image across image datasets with just the pupil circle showing, Figure 5.4.

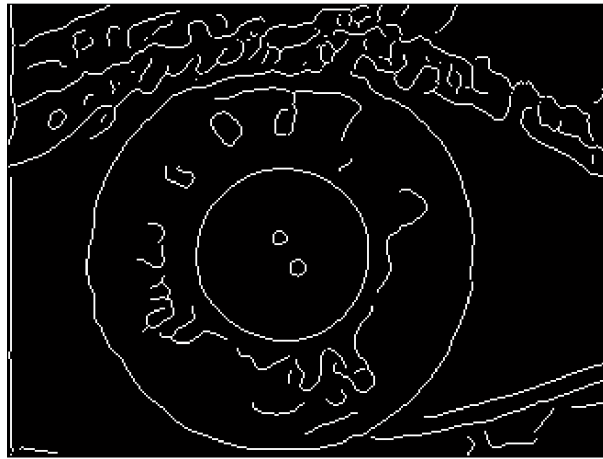


Figure 5.4 – MATLAB Canny edge detection of image 001/02 from the IITD image dataset

By logical bitwise AND'ing an enlarged Otsu thresholded eye image (Figure 5.3b) with the Canny edge map (Figure 5.4) the number of circles the Hough transform had to search for was dramatically reduced, and that the pupil circle was typically the strongest, Figure 5.5.

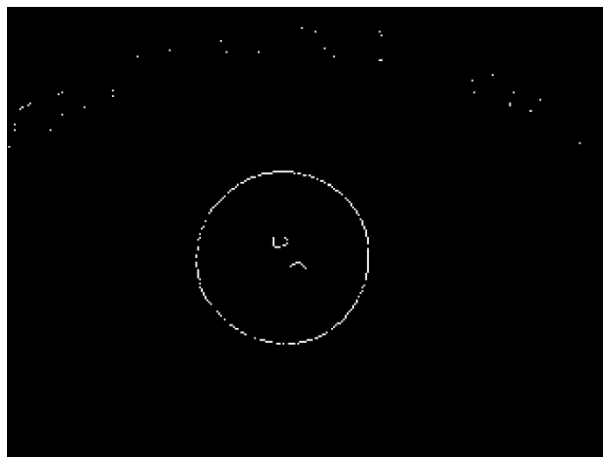


Figure 5.5 – Figure 5.4 logical AND'ed with Figure 5.3b

The Otsu thresholded image is enlarged by performing an image resize on the image, increasing its size in both the x and y directions by eight pixels. This adds four pixels to each side of the image that are cropped to return an image that is the same physical size as the original, but with the contents enlarged. The effect this has is to increase the size of the central disc and to move the eyelashes and other noise away from their actual location in the edge image.

Enlarging the Otsu thresholded image ensures that the pupil disc seen in Figure 5.3b intersects with the pupil ridge on the edge image in Figure 5.4 and moves much of the noise away from intersecting with noise lines on the edge image.

A Hough transform is then performed on the final edge image in Figure 5.5 and returns the pupil location. This was much faster than running the Hough transform on the edge image alone. Unfortunately, the varying lighting conditions across the image datasets once again showed the limitations of thresholding. The algorithm was having problems discerning the pupil consistently, occasionally getting the iris instead.

Figure 5.6 shows the fionar1 image from the MMU V1 image dataset with the threshold image produced by the first version of Segment Two. The images have been placed side by side such that a direct comparison of the locations included in the threshold can be seen.

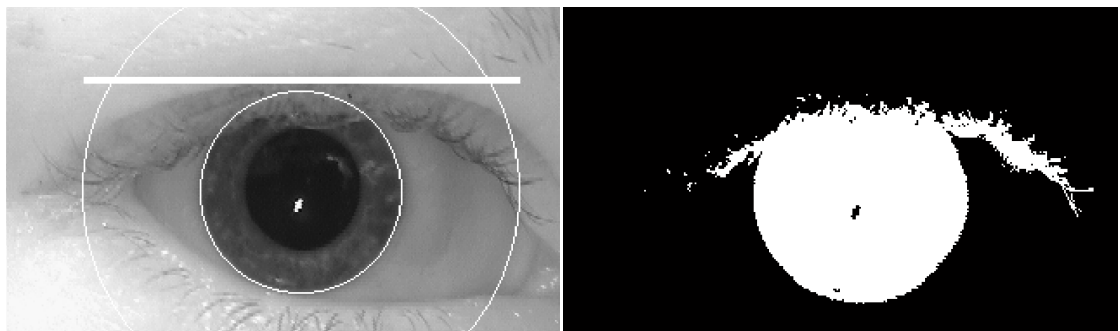


Figure 5.6 – The left image is fionar1 from the MMU V1 image dataset segmented with the first version of Segment Two, the right image is the monochrome threshold image.

Figure 5.7 shows the Canny edge image of fionar1 and the result of logical AND'ing the edge image with the threshold image in Figure 5.6.



Figure 5.7 – The left image is the Canny edge image of fionar1, the right image is the edge image logical AND'ed with the threshold image in Figure 5.6.

The iris and pupil boundaries are both clearly visible on the reduced edge image on the right in Figure 5.7. As it has a greater circumference, the iris needs more pixels for the edge line, so the Hough transform returns the iris circle instead of the pupil.

Manual evaluation of the segmented images showed that more than 25% of the 450 MMU V1 images had this problem. Literature on gamma correction [72, 150, 73] led to the consideration that gamma correction might hold part of the solution.

Pre-processing of the images had already intensity-stretched them to ensure the full range of intensities was used. Contrast had also been increased by mapping the intensity values in the original image such that 1% of the data at the lowest and highest intensities are removed.

A fixed gamma correction was added to the routine with the aim of increasing the separation between the iris and pupil pixel intensities. Several values of gamma were tested, and empirical evaluation of the results showed that a gamma correction value of 2.2 was the most effective across datasets. Applying this fixed gamma correction improved the segmentation results significantly, with less than 30 images still causing difficulties, but improvement was still needed.

Evaluating the final edge images from before and after gamma correction was applied showed a significant drop in the maximum and average number of non-zero points in the final edge image. The 'No Gamma' and 'Gamma' columns in Table 5.3 show a drop of approximately 50% for both average and maximum values. The 'Gamma & Adaptive' column will be discussed shortly.

Table 5.3 – Pixel count and averages for final edge image, without gamma correction, with fixed gamma correction and using gamma correction with adaptive intensity thresholding

	No Gamma	Gamma	Gamma & Adaptive
Average	560.34	285.44	115.42
Maximum	1520	754	287

Table 5.3 shows that once gamma correction was applied, less information was being presented in the final pupil boundary location edge image. When combined with the improved recognition results and the improvements seen in the manual inspection of the segmented images, this implied that the information that had been removed was noise.

This suggested that a further minimisation of the number of logical one's in the final edge image might improve the results even more. Any changes to the threshold needed to be image specific, so an adaptive intensity threshold was selected. The number of logical one's in the final edge image is counted, while repeatedly reducing the intensity threshold by 10%, until the number of bits has fallen to the desired level.

The final column of Table 5.3 shows the effect on the average and maximum number of logical one's in the final edge image when this adaptive intensity threshold is applied. The average was more than halved again and the maximum fell by two thirds. Manual evaluation of the segmented images showed that the image pupils were being correctly segmented.

Once the pupil location had been found consistently, a method of detecting the iris boundary was needed. The difference of sum-of-columns iris location method by Lin *et al.* [30] was still untested with an algorithm that could find the pupil reliably, so this was used as the iris detection method. Masek's eyelid detection was used to mark any occluding eyelid boundaries.

Testing was carried out with and without the adaptive intensity thresholding across all eight datasets. Curiously, the recognition results obtained were slightly worse when using the adaptive threshold, than when not using it. This was curious, as manual evaluation had conclusively shown the pupil segmentation to be improved with the adaptive threshold.

This highlights one of the challenges during early development of segmentation algorithms. Many image datasets are taken in consistent conditions, often on the same day or very close together and with the same image capture device with images that even appear to have been captured almost immediately after the last one.

This leads to an unfortunate situation, as seen with Lin *et al.* and Lu's algorithms in Chapter 4, where a particularly poor, but consistent, segmentation algorithm will locate

very similar locations on images of the same individual. More information is provided on this phenomenon in Appendix Appendix B.

The only solutions to this problem are to have the correct segmentation boundaries for each eye image available for comparison, or to manually inspect the segmented images. For these situations, manual evaluation is fairly effective as the segmentation boundaries are very clearly wrong, see Figure 5.6, but with 9,560 images to check it is a very slow and laborious task.

The results obtained when using the adaptive intensity threshold were carried forward for comparison as these had been demonstrated, via manual evaluation, to be based on a more accurate segmentation algorithm.

5.2.1 Initial Segment Two Results

Table 5.4 – Recognition results from Segment Two using Normalise One, Segmentation Mask One, Masek’s encode, Lin *et al.*’s iris boundary detection and Match One

Segment Two Recognition Results (Lin-Iris)				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	98.8%	53.4%	3.4	2.8%
CASIA V2D1	97.6%	16.1%	1.9	15.5%
CASIA V2D2	96.2%	4.2%	2.1	14.4%
CASIA V3i	97.9%	62.9%	3.3	3.5%
IITD	96.2%	-	3.3	4.4%
MMU V1	93.1%	2.2%	2.5	10.3%
MMU V2	53.9%	4.0%	0.0	35.9%
WVU Free	56.3%	7.5%	1.0	31.8%
Average	86.3%	21.5%	2.2	14.8%
Variance	44.9%	60.7%	3.4	33.1%

The results in Table 5.4 show that Segment Two using Lin *et al.*’s iris boundary detection and Masek’s eyelid detection outperformed every algorithm available at the time it was developed, when used across all the datasets.

The average true match rate of 86.3% is a good result that is better than the results from JIRRM at 60.9%, Antonino at 58.1% and Masek at 80.2%, but is lower than OSIRIS at 89.4%. The average separability results of 21.5% are again better than all except OSIRIS at 34.1%

Table 5.5 lists the code creation times attained using Segment Two with Normalise One, Match One and Masek’s encoding algorithm.

Table 5.5 – Code creation times of Segment Two using Normalise One, Segmentation Mask One and Masek’s encode

Segment Two Encoding Times (Lin Iris)		
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm:ss)
CASIA V1	0.47	00:05:55
CASIA V2D1	1.14	00:22:49
CASIA V2D2	1.16	00:23:18
CASIA V3i	0.01	00:23:02
IITD	0.58	00:21:45
MMU V1	0.71	00:05:21
MMU V2	0.45	00:07:27
WVU Free	1.1	00:01:28
Total		1:51:05

The times displayed in Table 5.5 were very fast, given the MATLAB environment. The time of 0.01 seconds per image for the CASIA V3i dataset was particularly fast. Using Segment Two instead of Masek’s segmentation algorithm had allowed the same eight datasets worth of images to be encoded in one hour 51 minutes, instead of the 36 hours 21 minutes taken by Masek’s algorithm’s.

Segment Two was 20 times faster than Masek’s segmentation method and more effective across all the datasets except the WVU Free dataset. The high encoding speed of this algorithm was due to the reduction in edge points used by the Hough transform for the pupil boundary and the use of Lin *et al.*’s iris boundary detection routine.

The iris boundary detection from Lin *et al.*, when used with a working pupil detection scheme, was moderately successful at finding the iris-sclera boundary. Figure 5.8 shows images from the MMU V1 dataset, which were segmented by Segment Two using Lin *et al.*’s iris boundary detection.

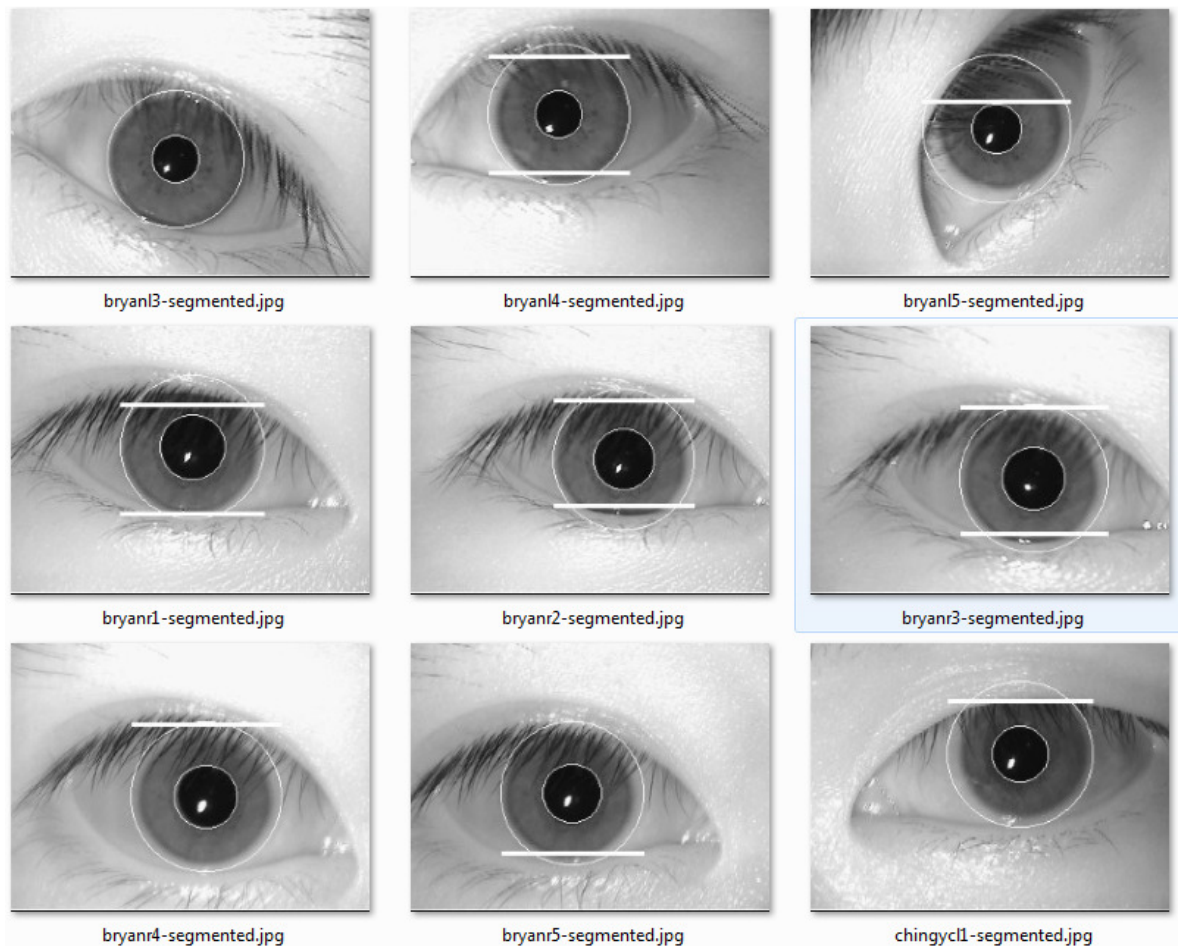


Figure 5.8 – Images from MMU V1 image dataset segmented with Segment Two using Lin *et al.*'s iris detection and Masek's eyelid detection.

The images in Figure 5.8 show that the iris is typically encompassed by the boundary line, but the line often does not follow the natural border between the iris and sclera, typically being oversized on at least one side.

The reason the boundaries are often oversized is that the sectors taken by Lin *et al.*'s algorithm often include eyelashes. The eyelashes are typically very dark, so Lin *et al.*'s method detects them as very strong edges that are often stronger than the iris boundary. The offset to one side or the other is due to the use of the pupil centre as the iris centre by Lin *et al.*, even though they are often not the same, as can be seen by the boundaries above.

5.2.2 Sum-of-Columns Method

The difference of sum-of-columns algorithm from Lin *et al.* was replaced, but the idea of taking sectors to locate the iris boundary was retained. This new algorithm also still used the idea of the sum of columns but dispensed with taking the difference.

The sum-of columns uses the information that the iris boundary is a regular shape and the sectors that are sampled are quite small. By adding up the columns of each sector the intensity value differences between columns are amplified where the values in a column are similar and reduced where the values are dissimilar.

Therefore, columns with mostly iris will add up to a lower value than columns with mostly sclera. At the point where the sector changes from iris to sclera the sums of the columns will have an increased gradient than the individual values in the sector. Lin *et al.*'s method calculates the differences between each of these column sums to arrive at the gradient.

The first change from Lin *et al.*'s method was to take the iris sectors from 10° below horizontal down a further 25° ; this moves them out of the range of the eyelashes on many images and means that when the eyelashes are included they often span columns, thus reducing their impact. The 10° angle was derived empirically, by gradually increasing the start angle used from 0° to 10° in 2° increments and evaluating the results across the eight datasets.

The outcome showed segmentation performance that only varied the true match result by 0.1%, with no measurable difference in separability, decidability and EER, for start angles between 10° and 15° . These results were obtained using Lin *et al.*'s 20° swathe, so the start angle was selected as 10° and the swathe increased to 25° to improve the accuracy of the sum-of-columns. This combination proved to be the most effective at maximising data, and minimising noise.

The second change from the algorithm from Lin *et al.* was that the minimum iris radius was increased to be a percentage of the pupil rather than just a fixed increase. Figure 5.9 shows the location of the two sectors on image 0002_002 from CASIA V2D1. The vertical white lines indicate the detected iris diameter.

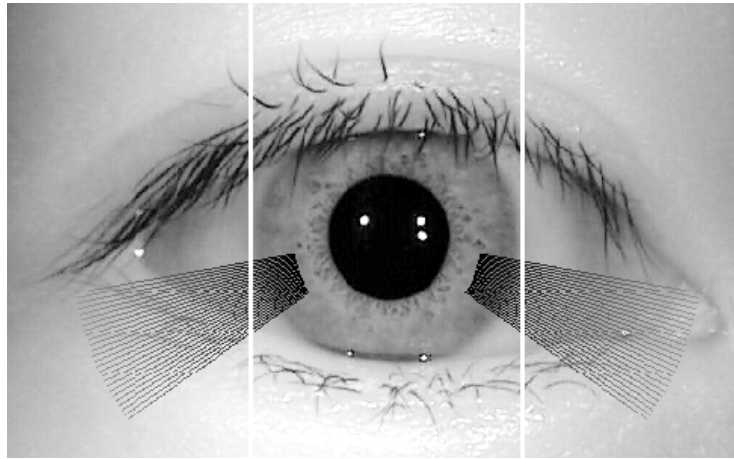


Figure 5.9 – Image 0002_002 from CASIA V2D1 gamma corrected with boundaries & sectors marked on it.

Once the sectors in Figure 5.9 are extracted from the image, the contrast is adjusted, a linear intensity stretch is applied and finally a 7x7 Gaussian blur [70] is applied. The sectors are shown in Figure 5.10 and were taken from the sector locations shown on Figure 5.9.



Figure 5.10 – Original iris sectors from Figure 5.9

Figure 5.11 shows the effect of intensity stretching the original sectors in Figure 5.10

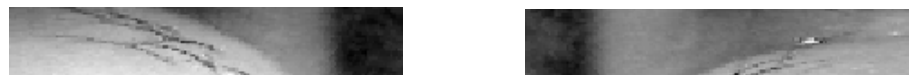


Figure 5.11 – Intensity stretched Figure 5.10 images

The contrast is significantly increased in Figure 5.11 as compared with Figure 5.10. Figure 5.12 shows the final Gaussian smoothed sectors.



Figure 5.12 – Gaussian smoothed Figure 5.11 images

After the intensity stretching shown in Figure 5.11, the iris area becomes very dark in comparison to the rest of the sector, but so do the eyelashes. The Gaussian smoothing blurs the image such that the eyelashes and other small features almost disappear. This removes

the small details that confuse the edge detection. Since these sectors are only used for finding the iris boundary, and are not used as part of the encoding process, this loss of detail is in fact desirable.

The columns are then summed creating the two column vectors shown in Figure 5.13. The last trough in Figure 5.13a and the first peak in Figure 5.13b indicate the iris boundaries.

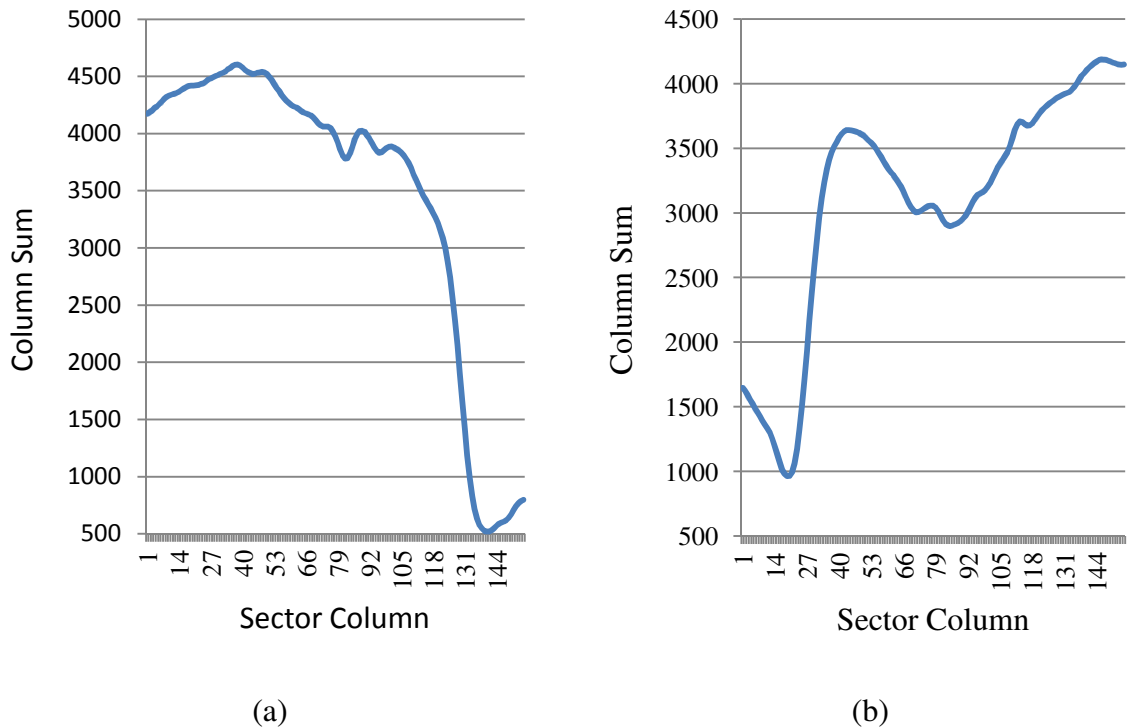


Figure 5.13 – Sums of columns for both the left (a) and right (b) sectors respectively. Y-axis indicates sum, x-axis indicates x position within the original sector

The same horizontal mask that is used in the Sobel edge detection method is convolved with the left and right sum of columns, as shown in Figure 5.14. The largest trough in Figure 5.14a, and the largest peak in Figure 5.14b, indicate the iris boundaries.

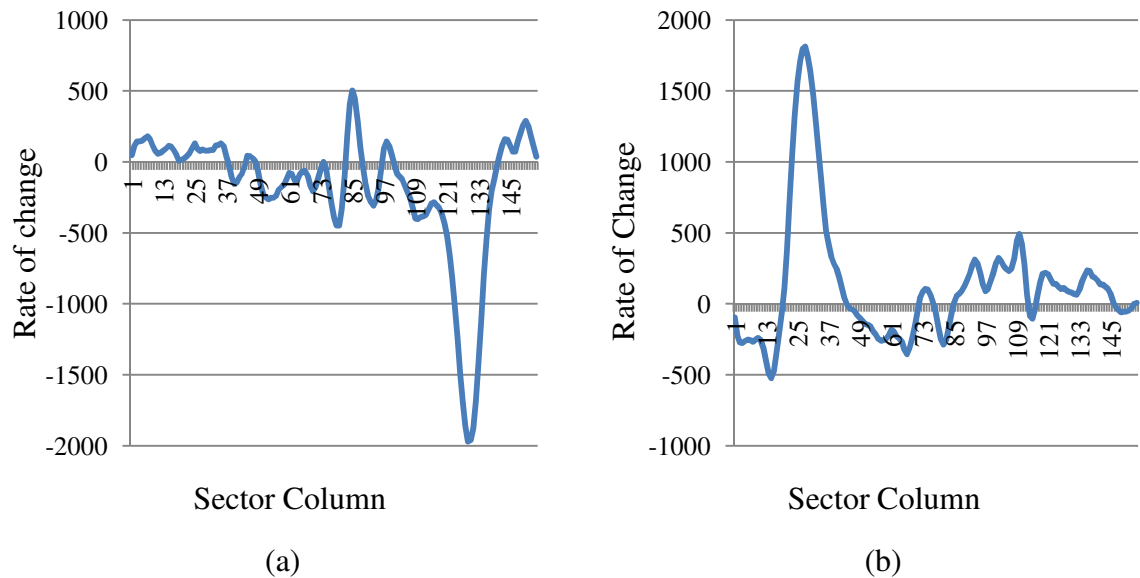


Figure 5.14 – The gradients of the column sums for both the left (a) and right (b) sectors.

Y-axis indicates gradient, x-axis indicates x position within the original sector

The minimum value in Figure 5.14a and the maximum value in Figure 5.14b are then found and the left and right boundaries are calculated. These boundaries are used with the pupils x centre to find the x offset of the iris in relation to the pupil and the iris centre is adjusted accordingly.

Figure 5.15 shows the final image with all segmentation boundaries present.

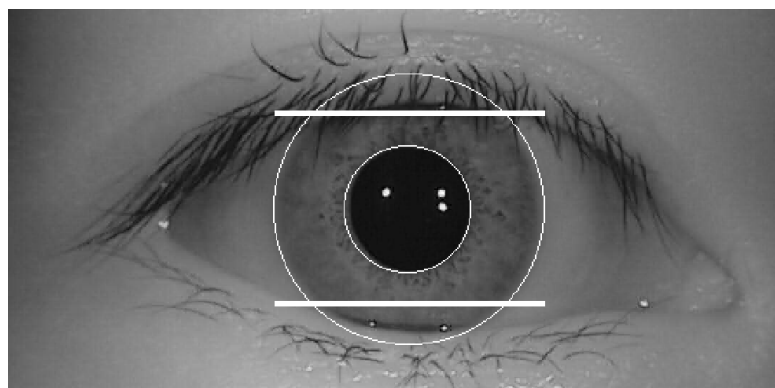


Figure 5.15 – CASIA V2D1 image 0002_002, segmented using Segment Two

Image 0002_002 from the CASIA V2D1 dataset in Figure 5.15 shows the circular iris-pupil and iris-sclera boundaries and the linear top and bottom eyelid boundaries. The iris-sclera boundary in Figure 5.15 is more accurate than achieved with Lin *et al.*'s difference-of-sum-of-columns method.

5.2.3 Final Segment Two Results

Table 5.6 presents the Recognition results from using Segment Two with the iris boundary detection described. Normalise One, Segmentation Mask One, Match One and Masek's encoding algorithm were used for the remainder of the flow.

Table 5.6 – Recognition results from Segment Two using the described iris boundary detection, Normalise One, Segmentation Mask One, Masek's encode and Match One

Segment Two Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	99.3%	70.2%	3.7	2.3%
CASIA V2D1	95.8%	2.0%	1.8	17.2%
CASIA V2D2	89.8%	6.4%	1.7	20.1%
CASIA V3i	97.7%	65.8%	3.1	4.1%
IITD	96.6%	-	3.5	3.2%
MMU V1	97.6%	3.8%	3.3	5.4%
MMU V2	64.2%	0.0%	0.0	28.3%
WVU Free	65.0%	2.5%	1.1	24.0%
Average	88.3%	21.5%	2.3	13.1%
Variance	35.1%	70.2%	3.7	26.1%

Recognition rates are significantly higher with this configuration and exceed Masek's algorithm's results for all eight image datasets, except for separability, which is lower. The average true match rate across the datasets used was 88.3% versus Masek's 80.2%, the average EER was 13.1% versus Masek's 19.2% and the average decidability was 2.3 versus Masek's 2.0. The average separability remained unchanged from using Lin *et al.*'s iris-sclera detection at 21.5%.

Only OSIRIS obtains better true match results with an average rate of 89.4% as well as higher separability at 34.1%. However, this algorithm beats OSIRIS's 2.2 average decidability and 13.6% average EER.

5.2.4 Conclusions

This section looked at the development of a new thresholding method for reducing the number of points in an edge image during pupil detection. When combined with the Hough transform this proved to be very effective at finding the pupil boundary. The outcome was improved, when compared to that obtained using Masek's algorithm, on all datasets except the WVU dataset

This section also re-used the iris-sclera detection from Lin *et al.*, finally pairing it with a working technique for pupil detection. The fact that it could be seen that Lin *et al.*'s iris-sclera detection could be reasonably effective, when the iris-pupil detection was so ineffective, highlighted the benefits of using the generic platform.

Replacing the iris boundary detection from Lin *et al.* with the described sum-of-columns based iris detection improved recognition rates significantly, with no visible impact on encoding times. While the alterations to the iris-sclera detection algorithm were relatively small, they were very effective.

The good results attained by the early versions of this algorithm were once again due to very consistent images within datasets, strengthening the case for an automatic segmentation evaluation routine.

It was shown that it is possible to use intensity thresholding as part of an effective recognition algorithm that works on multiple image datasets, but that this needs to be adaptive in order to remain effective.

Finally, the use of the Hough transform, even within the generic platform environment was demonstrated to be effective and fast, so long as the points within the input edge image are sufficiently minimised.

The next section describes the development of Segment Three

5.3 Segment Three

This section describes a novel segmentation method using a simple but novel circle bisector and a sum-of-columns methods. As with Segment Two, Segment Three uses a number of techniques to pre-process the eye images. Each image is intensity stretched, gamma corrected and then intensity normalised [70]. This adjusted image is then used as the input to all four boundary detections (Pupil, iris, top and bottom eyelids).

The Canny edge detection is used on the adjusted image to find the edges and then the developed circle bisector method is used to find the pupil and eyelids. The sum-of-columns method is used for the iris-sclera boundary detection

The circle bisector method was developed to replace the Hough transform [41], which tends to be very good at finding circles in an image but it is quite slow. Two other detractors for the Hough transform are its large memory requirements and an inability to identify circles with centres outside the boundaries of its accumulator. Unless the Hough accumulator is sufficiently large, this causes a problem for eyelids, whose centres often fall outside the image boundaries.

The solution selected to resolve both Hough weaknesses was the circle bisector method. The circle bisector method is discussed in more detail in Appendix Appendix A. Lu's [60] circle bisector method was evaluated in Section 4.5.2 and shown to be flawed. However, the thresholding used by both Lu and Lin was thought to be the major cause of this and a viable threshold method had been developed for Segment Two which it was thought might correct the problems seen previously with Lu's method.

Given a list of points in an edge image, the difficulty arises from knowing *which* points are on the same circle. The other problem when using the circle bisector method in a computer, as discussed in Appendix Appendix A, is that the boundary points have been digitised, and this quantisation of the data leads to errors when it is used in calculations.

Lu's solution to these problems was to repeatedly pick random edge points and then use standard deviation to find the most common. This worked when the significant proportion of points on the edge image were from the boundary being sought, but failed as the proportion of non-pupil circle points increased.

The uniqueness of Segment Three is the way in which the points passed to the circle bisector method are selected. When an eye image is passed through Canny edge detection the resultant edge image comprises a binary image with many discontinuous edge sections. The pixels in each section are locally related, allowing the assumption that any circles found can be considered to have a higher likelihood of being genuine circles.

The proposed method searches the image for an edge point and when it finds one, it uses a recursive boundary search to find all the points that are attached to the first. Figure 5.16 shows an example of how this works; the '1's represent edge points, the '0's non edge points.

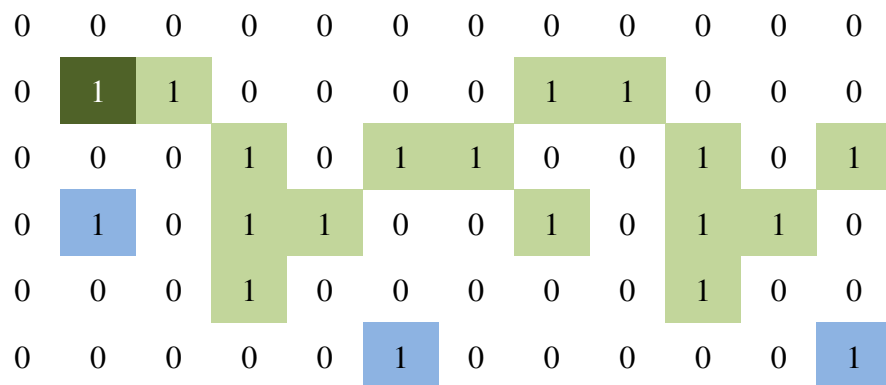


Figure 5.16 – Example edge image showing the connected and unconnected edge locations

The method starts from the top left corner and moves from left to right until it finds a '1', wrapping round to the start of the next line at the end of each line. In the example shown, the dark green square with the white '1' would be the first edge location discovered. The method calls the boundary scan with the x,y location of the discovered '1'.

The boundary scan stores the current location and then looks at the eight surrounding squares for any '1's. For each '1' found in those eight locations the boundary scan calls itself with the location of those '1's. Each call adds the location it is called with to the list of connected boundary points. When the boundary scan finishes it returns a list of all the connected boundary points, the green squares in Figure 5.16. The blue squares are not adjacent so are not returned.

The final action is to clear all the points that have been found from the edge image, so that they are not found on the next search. In this way Segment Three is able to find groups of edge points that are typically related to the same feature within the original image.

Checking every point against every other point would be incredibly slow – three hours in a test for one image, and quantisation noise prevents consecutive points being useful. Therefore, Segment Three takes the natural contiguous groupings of edge pixels produced by the edge detection and divides the points within each group into four equal parts with a coordinate taken from each of the first three quartiles. This ensures that quantisation noise is minimised for groups of more than 20 pixels.

The three co-ordinates are passed to the circle bisector algorithm which uses the circle bisector method to calculate the centre and radius of the circle that intersects all three points. If a circle is discovered and is already in the list then the count for that circle is incremented, otherwise it is added to the end of the list. The pointers are incremented to the next pixel in their quartile on each iteration of the loop until the third pointer is at the last point in the list.

Having collected a list of circles, Segment Three tries to find similar circles in order to overcome the quantisation, real life irregularities and edge detection noise. This is achieved by averaging circles with all three parameters (x, y & r) within 3% of each other, thus reducing the effect of circle digitisation and strengthening the count for actual circles present.

After the number of circles has been minimised each circle is analysed to see what percentage of the circle intersects with points on the original edge image, and this percentage is then added to the circle count.

The iris is located using the sum-of-columns method from Segment Two, but with two additional Gaussian smooth operations. It was found that applying the Gaussian smooth operation three times consecutively led to a further reduction in the impact of noise such as eyelashes without blurring the iris-sclera boundary beyond usability.

The top and bottom eyelids use the same circle bisector method used to find the pupil, but they use a reduced edge image which is taken from directly above the pupil centre for the

top eyelid and directly below the pupil centre for the bottom eyelid. This reduces the number of points considered and focuses on the likely locations of the eyelids.

The circle bisector is not constrained by the size of the accumulator matrix or by predetermined circle radii, as it would be for the Hough transform. This allows it to find the much bigger eyelid circles and their centres without the need for an excessively large accumulator, thus increasing the accuracy of the eyelid boundaries returned.

5.3.1 Segment Three Results

Table 5.7 – Recognition results achieved with Segment Three using Normalise One, Segmentation Mask One and Match One with Masek’s encoding algorithm

Segment Three Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	99.9%	27.2%	4.4	1.3%
CASIA V2D1	99.6%	47.4%	3.0	5.3%
CASIA V2D2	92.3%	34.6%	2.2	14.0%
CASIA V3i	99.0%	0.1%	5.0	0.6%
IITD	96.9%	-	5.1	2.1%
MMU V1	94.0%	7.4%	2.4	15.2%
MMU V2	91.3%	0.6%	2.2	13.5%
WVU Free	91.3%	12.5%	1.4	15.1%
Average	95.5%	18.5%	3.2	8.4%
Variance	8.6%	47.3%	3.7	14.6%

The recognition results for Segment Three were significantly better than results seen previously. The average true match rate was 95.5% with three datasets returning 99% or better. Average decidability was 3.2 and average EER was 8.4%, both of which were a significant improvement over Masek’s code and over the previous segmentation algorithms created by this work.

The only metric that was disappointing was separability at 18.5% which was lower than both OSIRIS and Segment Two achieved. However, the combination of 18.5% separability with 3.2 decidability and 8.4% EER showed that this algorithm was the best yet seen for determining between authentic and imposters.

Figure 5.17 directly compares the results obtained with OSIRIS against those obtained with Segment One, Segment Two, Masek, OSIRIS and Li’s segmentation algorithms.

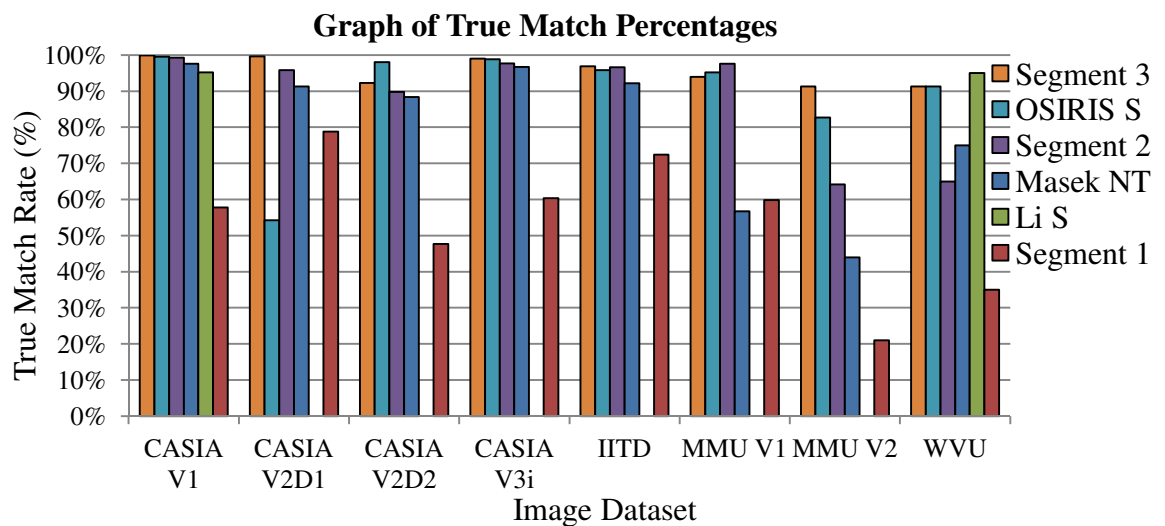


Figure 5.17 – Comparison of true match results from Segment Three against the true match results attained by Segment One, Segment Two, Masek, OSIRIS and Li’s segmentation algorithms.

Figure 5.17 shows that Segment Three clearly outperforms the best of the algorithms presented so far for the CASIA V1, CASIA V2D1, CASIA V3i, IITD and MMU V2 datasets. Only OSIRIS and Segment Two are able to return better results on the remaining three datasets.

Table 5.8 lists the encoding times attained using Segment Three with Normalise One and Masek’s encoding algorithm.

Table 5.8 – Code creation times attained using Segment Three with Normalise One and Masek’s encoding algorithm

Segment Three Encoding Times		
Dataset	Average Code Creation Time (s)	Code Creation Time (h:mm)
CASIA V1	0.6	0:08
CASIA V2D1	1.0	0:19
CASIA V2D2	1.2	0:24
CASIA V3i	0.5	0:22
IITD	0.5	0:20
MMU V1	0.4	0:03
MMU V2	0.5	0:07
WVU Free	1.3	0:02
Total		1:45

The code creation times in Table 5.8 are on a par with the times from Segment Two. Even though the CASIA V3i does not manage the very low result of 0.01 seconds achieved by Segment Two, the per image code creation times for Segment Three are much more consistent. An average code creation time for all the datasets of one hour 45 minutes is very similar to the one hour 51 minutes from Segment Two, so 20 times faster than Masek’s algorithm.

5.3.2 Conclusions

This section proposed a way of using the circle bisector method that is different from that in current literature. It uses intensity adjustment, followed by Canny edge detection to identify edges in the iris image. These boundaries are then processed using the circle bisector method to find the pupil and eyelid boundaries. The iris is located using the developed sum of columns methodology.

Segment Three outperforms all the other algorithms and methods tested in the text. The circle bisector method developed was very successful at finding the pupil circle and improved recognition results when used as the eyelid boundary detector instead of Masek's line eyelid detection.

Segment Three marks the culmination of all the segmentation knowledge gained. It builds strongly on the knowledge gained from Segment One and Segment Two, as well as the investigations carried with the open-source algorithms and methods from the literature.

This algorithm also demonstrates that effective use of an underlying principle can be as important as the principle itself. The sum of columns function is used for iris recognition by many researchers, including Lin *et al.* [30], Mesecan *et al.* [151] and Ko *et al.* [152], as well as in the work carried out for this research. It is not a new methodology, but the differences in the way it was used by Segment Two and Three when compared to the algorithm from Lin *et al.* meant that the results were much improved.

The same is true of the circle bisector method, which is a well understood method within the field of mathematics for finding circle centres and radii. Yet this well known and understood method is not used within the field of iris recognition, except by Lu [60]. The most likely explanations for this are the popularity of the Hough transform and the challenge with overcoming the errors caused by quantisation noise in the circumference points. Yet Segment Three demonstrates that the circle bisector is a very fast and effective method when given correctly spaced out points that are related to each other.

The next section looks at an implementation of automatic segmentation evaluation that was proven to be effective

5.4 Automatic Segmentation Evaluation

Several times in the text, it has been stated that consistent images can allow a poor segmentation algorithm to present better overall results than it actually achieved. As stated in Appendix Appendix B, a consistent image is one that is of the same individual taken in very similar circumstances.

The result of this is that a poor, but consistent, segmentation algorithm will select a similar part of the image and the following processes will find sufficient consistency to match images correctly, despite no part of the iris being included in the unwrapped data.

One solution to this problem is the manual evaluation of segmented images output by the generic platform during the automatic algorithm processing. This was used to good effect several times during this research but it is very time consuming and extremely tedious.

When an algorithm is very poor, manual evaluation can be completed quickly, as the segmentation is visibly very wrong. However, this is far from scientific and does not show in the output results at all.

An automated segmentation evaluation was proposed to be part of the platform. However, in order to do this the platform would need to know what constitutes a 'good' segmentation and what constitutes a 'bad' segmentation. The obvious way would be to provide the platform with known 'good' segmentation boundaries, but, with thousands of images and four different boundary types – lines, circles, ellipses and splines – manually entering these was a daunting task.

Another, more practical solution presented itself, in the form of the results from all the tests carried out on different algorithms. Each test produced a pre-calculation database containing all the segmentation boundaries calculated by those algorithms.

These pre-calculated results could be used to calculate the standard deviation of each boundary parameter; this standard deviation result could then be used to automatically score new segmentation results. There is, however, a problem with this approach - if poor segmentation algorithms produced more than 50% of the reference datasets then the scoring will also be wrong.

The solution to this was to add the ability for humans to grade boundaries, the algorithm would score them based on the standard deviation and then human operators could manually grade these as required or necessary.

As many of the results had been manually verified during testing, it was felt that the results available were sufficiently good to allow this method to be used.

A separate script provided with the generic platform, 'CreateReferenceDBs', reads the pre-calculated databases, extracts, and combines their data into a single reference database for each image dataset. Once the boundaries are added, the script then finds the standard deviation and scores each boundary from every algorithm for each image and they are sorted into order.

Once the reference datasets are created they are available during analyses output. During the output of an analysis, if a reference database is available then the segmentation boundaries from the analysis results will be compared with the reference database and given a score based on their standard deviation from the mean.

The standard deviation is calculated for each component on the boundary, x , y and radius for a circle, and the lowest standard deviation returned is the score that is used for that boundary. A graph showing a distribution of the standard deviations, normalised to 100, for each of the four boundaries, pupil, iris, top eyelid and bottom eyelid is output – An example of this is Figure 5.19 in the 'Automatic Segmentation Results' section. Average figures are also included in the output text file.

The reference datasets are also available from the results pane within the generic platform, shown in Figure 5.18.

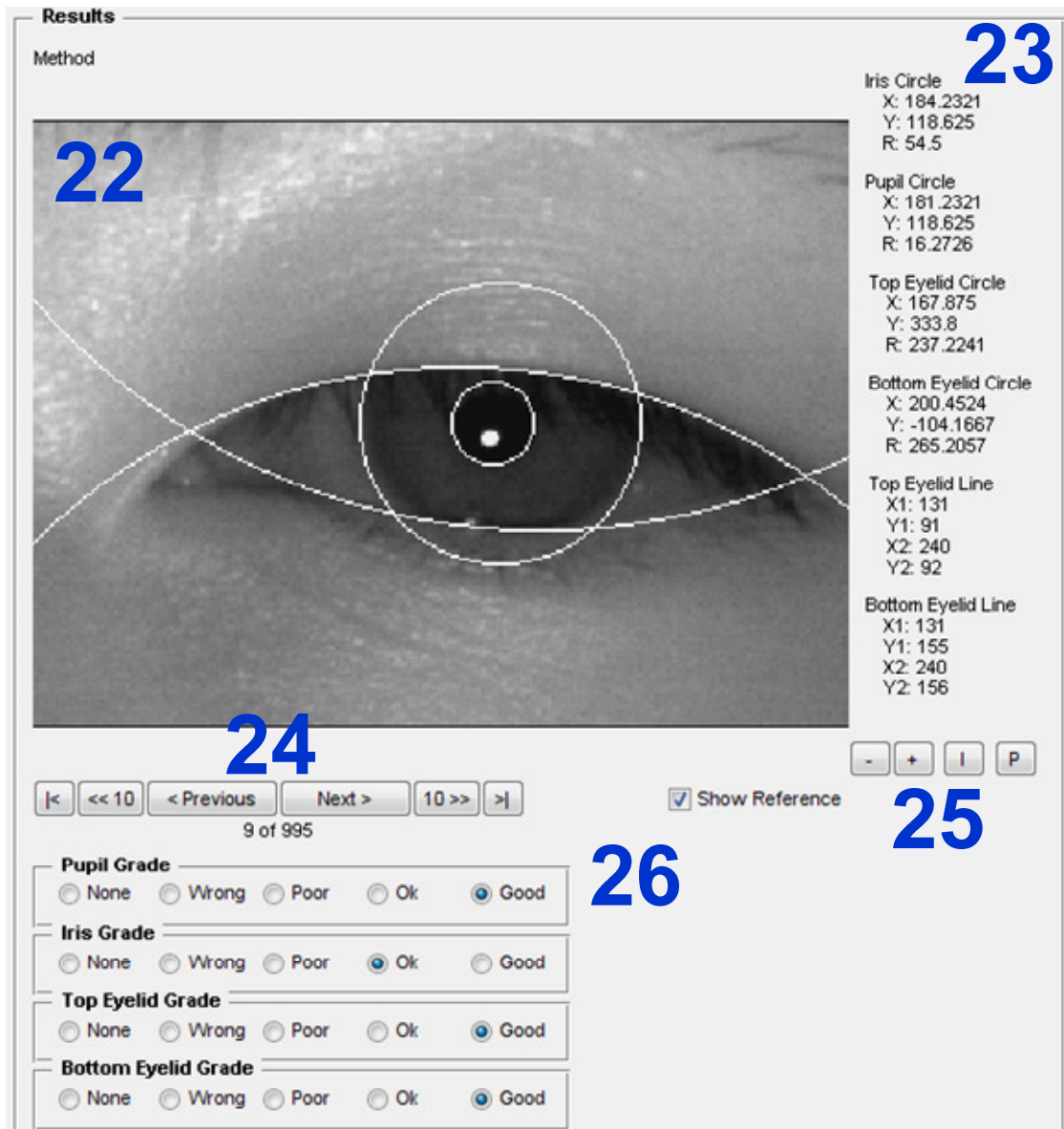


Figure 5.18 – Generic platform results pane showing the reference segmentation for image 010204 from the MMU V2 image dataset.

The results pane allows the user to switch to using the reference boundaries so that they may check and grade them. This mode is enabled with the ‘Show Reference’ checkbox in Figure 5.18 near the reference grading controls (26). The iris image with segmentation boundaries applied is at (22), with the coordinates for each of the boundaries listed in (23). The controls at (25) allow the image to be zoomed and the image can be selected using the controls at (24).

5.4.1 Automatic Segmentation Results

Figure 5.19 shows the automatic segmentation results for Segment Three used with the CASIA V2D1 image dataset.

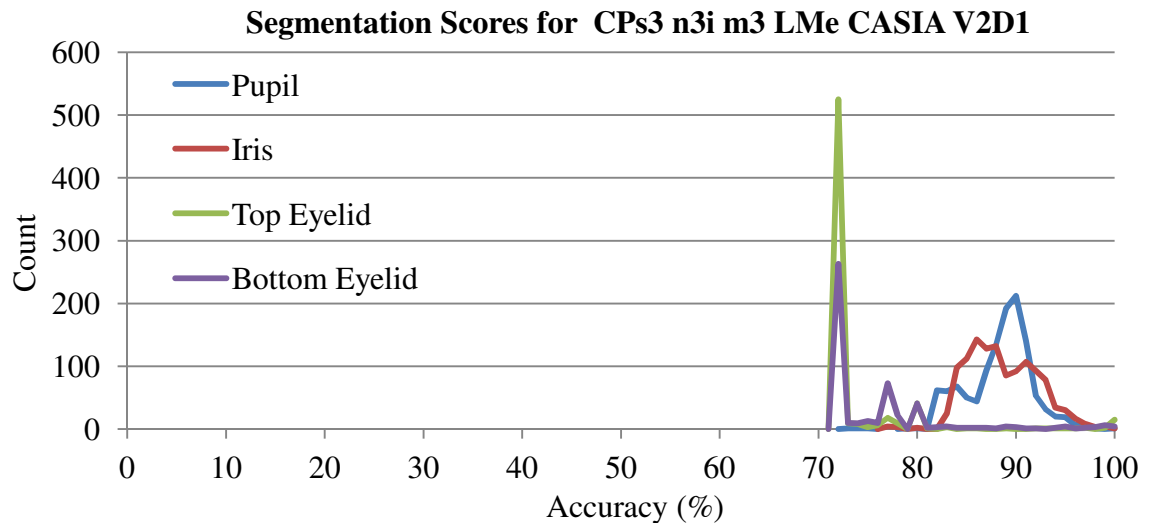


Figure 5.19 – Automatic segmentation analysis for Segment Three with the CASIA V2D1 image dataset.

The results in Figure 5.19 show what would be expected for a competent algorithm - all the results are grouped at the far right of the graph indicating a high score. Figure 5.20 shows the automatic segmentation results for Lin *et al.*'s algorithm used with the CASIA V2D1 image dataset.

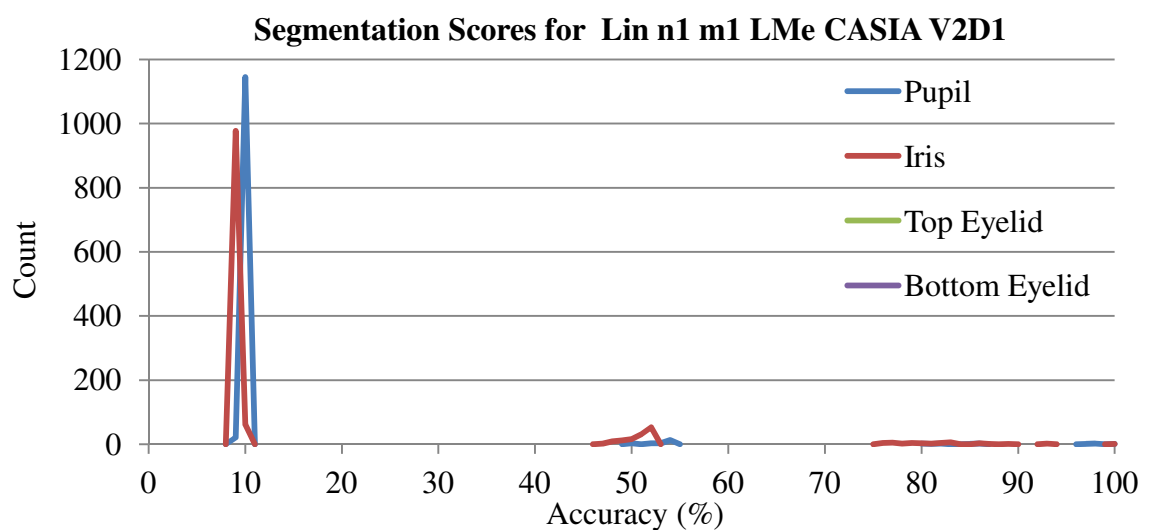


Figure 5.20 – Automatic segmentation analysis for Lin *et al.*'s segmentation method with the CASIA V2D1 image dataset.

The results in Figure 5.20 show what would be expected for a poor algorithm with many of the results grouped at the far left of the graph, indicating a low score. The text file report confirms the poor result for Lin *et al.*'s algorithm when used with the CASIA V2D1. The text file states the percentage of boundaries detected that match the current best boundary location for each eye image:

```
Segmentation:
  Pupil:          10.3%
  Iris:           15.1%
  Top Eyelid:    48.4%
  Bottom Eyelid: -
  Average:       24.6%
```

The results from Lin *et al.*'s algorithm are clearly very poor with only the top eyelid results performing well. Comparing these results with those from Segment Three, the difference is very clear:

```
Segmentation:
  Pupil:          84.0%
  Iris:           83.0%
  Top Eyelid:    76.4%
  Bottom Eyelid: 76.9%
  Average:       80.6%
```

These results show a strong algorithm with consistently strong segmentation results. This then is a useful indicator to the performance of a segmentation algorithm. These results were consistent with the manual evaluation results obtained during the course of the research.

5.4.2 Conclusions

This section proposed a novel method of automatically analysing segmentation results. The method was shown to be successful in highlighting poor segmentation performance. This provides a useful measure of the performance of a segmentation algorithm.

The method uses the previously obtained segmentation results in the pre-calculation databases and uses statistical analysis to determine the most popular results for a each boundary on a given iris image. Unfortunately, there is no real certainty that the pre-calculated results to from which the 'best boundaries' are being selected are 'good' and the only way to be sure is to manually inspect the reference values.

The use of manual grading of the segmentation boundaries in the reference database allows for a degree of confidence in the boundaries being used for comparison. However, this is a significant task and, though progress has been made, more is needed. It is also important to get additional, differently biased, opinions on the quality of the segmentation boundaries, as what may appear to be good to some viewers, may appear only adequate, or even poor, to others.

For these reasons the results of this process were not used in the text, as they are still subjective and at best indicative of good and poor results. It is hoped that when the code is released generally, additional individuals will be willing to contribute to the task of grading the boundaries, so lending more confidence to the scores returned.

The next section discusses Normalise One which is a normalisation algorithm designed to add the ability to cope with circular and elliptical eyelid boundaries but have no measureable effect on the results output

5.5 Normalise One

Normalise One was created to replicate Masek's normalisation code but to extend it to cope with circular and elliptical eyelid boundaries, and to use Segmentation Mask One (SM1) as its default segmentation mask (Section 5.7). SM1 removes the thresholding present in Masek's segmentation mask and adds the ability to mask elliptical and circular eyelid boundaries.

Masek's normalisation routine only works with circular boundaries for the iris, so a normalisation routine that returned the same results but was capable of unwrapping ellipses was necessary to enable testing of segmentation algorithms that out elliptical boundaries [12]. As Masek's routine completed in 0.01 seconds there was no clear need to increase speed.

Normalise One with SM1 would allow segmentation routines that produced elliptical boundaries to be tested using Masek's flow, so that segmentation routines could be compared against each other.

Figure 5.21 shows the basic theory of Daugman's iris unwrapping technique. Once the iris boundaries at the pupil and sclera have been identified the area between is sampled and placed in a rectangular array. Linear interpolation is used to calculate any missing values during the sampling. Linear interpolation calculates the average value of the values either side of the missing value and uses that average for the missing value.

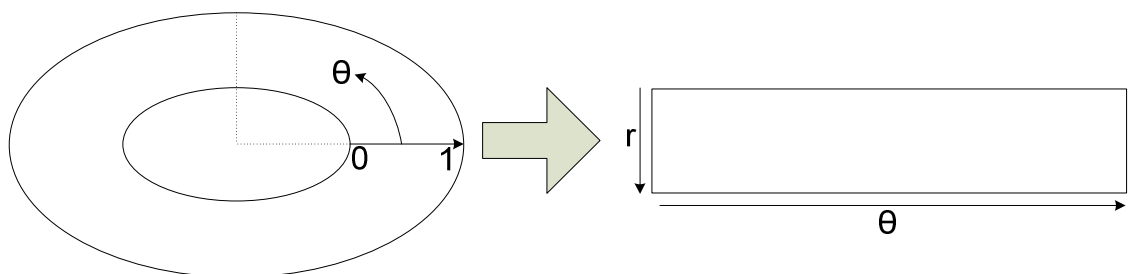


Figure 5.21 – Daugman's rubber sheet model[77], translating an elliptic iris from Cartesian to polar coordinates

Pythagoras theorem is used to calculate the x,y location of each circumference point for both the pupil and sclera boundaries and then the points are sampled evenly spaced along the line between the two boundaries.

As Figure 5.22 shows, these circular boundaries often do not have their centres in the same place, so are not concentric.

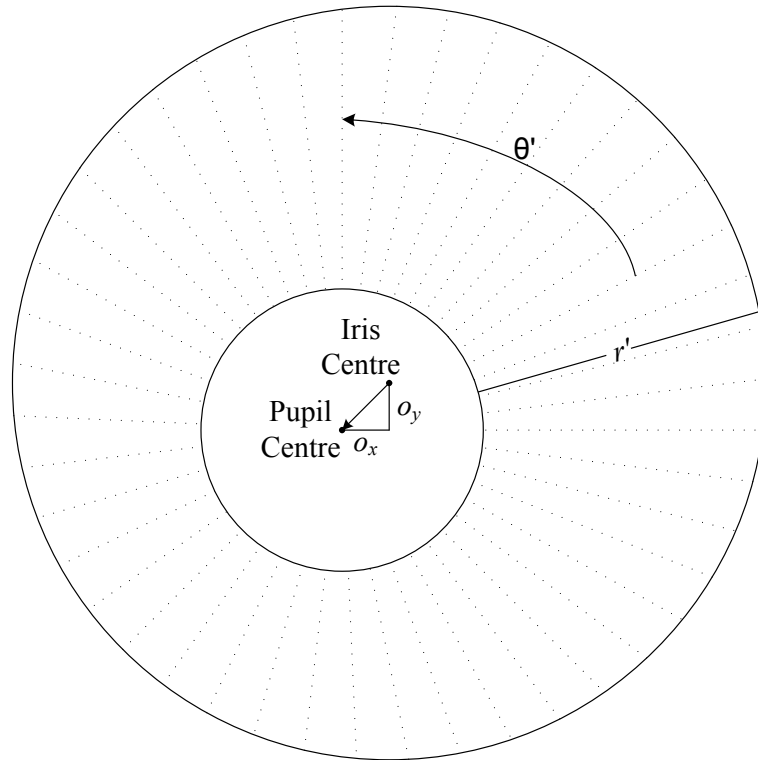


Figure 5.22 – Daugman’s rubber sheet model[77, 31] providing pupil displacement correction showing exaggerated pupil displacement for illustration

It can be seen on Figure 5.22 that the radius between the iris and pupil varies with θ , so a remapping function is needed so that the sample points at different values of θ around the circle are re-scaled. Thus, the radii for each ellipse must be calculated for each value of theta using Equation 5.1, which shows Masek’s solution to this.

$$r' = \sqrt{\alpha} \beta \pm \sqrt{\alpha \beta^2 - \alpha - r_i^2} \quad 5.1$$

where: $\alpha = O_x^2 + O_y^2$

$$\beta = \cos\left(\pi - \arctan\left(\frac{O_y}{O_x}\right) - \theta\right)$$

$$O_x = PupilX - IrisX$$

$$O_y = PupilY - IrisY$$

θ is the angle around the circle

r_i = Iris radius

O_x and O_y are the delta between the pupil and iris origins in pixels along the x and y axes respectively. α is the sum of the squares of the x and y deltas. β is the angle around the circle that corrects for the offset in circle centres. r' is the radius at the selected angle of θ which ranges from 0 to 2π radians.

Equation 5.1 was modified such that the single iris radius, r_i , was replaced with r_θ . To do this the circumference points at each angle of theta were calculated using the standard equations, see equations 5.2 and 5.3. The radii at each value of theta was then calculated using Pythagoras again, see equation 5.4.

$$x_t = x_o + r_a \times \cos \theta \cos \varphi - r_b \times \sin \theta \sin \varphi \quad 5.2$$

$$y_t = y_o + r_a \times \cos \theta \sin \varphi - r_b \times \sin \theta \cos \varphi \quad 5.3$$

$$r_\theta = \sqrt{x_t^2 + y_t^2} \quad 5.4$$

where: θ is the angle around the ellipse

φ is the angle between the x-axis and the ellipse's major axis

r_a is the ellipse's major axis

r_b is the ellipse's minor axis

x_o and y_o are the coordinates of the ellipse centre

x_t and y_t are the deltas in the x and y directions between the centre and the circumference for the given angles of θ and φ

r_θ Radius of the ellipse for the given angles of θ and φ

r_θ was then replaced r_i in equation 5.1 to give equation 5.5.

$$r' = \sqrt{\alpha} \beta \pm \sqrt{\alpha \beta^2 - \alpha - r_\theta^2} \quad 5.5$$

where: $\alpha = O_x^2 + O_y^2$

$$\beta = \cos \left(\pi - \arctan \left(\frac{O_y}{O_x} \right) - \theta \right)$$

$$O_x = PupilX - IrisX$$

$$O_y = PupilY - IrisY$$

θ is the angle around the ellipse

r_θ Radius of the ellipse for the given angles of θ and φ

Normalise one was tested against those datasets that were selected as most useful for testing with Masek's segmentation, encoding and matching algorithms. The recognition results produced by the generic platform outputs obtained with Masek's normalisation routine when compared with Normalise One had no measurable differences. There was also no measurable difference between the execution times, within the ability of the generic platform to measure them.

Normalise One allowed Li's segmentation algorithm, Section 4.3.2, which produces ellipses for the iris boundaries, to be tested with Masek's encoding algorithm.

5.6 Improving Recognition Performance During Normalisation

In biometrics, normalisation is the process of converting sample data to a form that is comparable with all the other samples. For iris recognition this typically takes two forms, unwrapping the iris to a rectangular shape, as per Daugman [2, 31, 28] or leaving the iris as a ring as Birgale and Kokare [83] and Boles do [90].

In both cases the image pixels that do not form part of the iris are removed and the resulting normalised iris is typically altered to be a consistent size. In unwrapping, this is achieved by sampling across the radius and around the circumference. When left as a ring, this is achieved via re-sizing the image.

As part of the normalisation process, an additional step of segmentation masking can occur [31, 28]. This involves marking on the image any pixels that are outside the segmentation boundaries, typically by setting them to something detectable - in the generic platform this is done using the NaN (not a number) value. This is only carried out for the top and bottom eyelids but means that if they are detected correctly, and are partly obscuring the iris, then they will not be mistaken for iris texture, thus ensuring that they do not affect the results.

Normalise One and Segmentation Mask One were developed as part of this work. These methods present nothing novel, but they increased the functionality within the generic platform by adding support for ellipses and removing the fixed thresholding in Masek's code. Normalisation as a process is typically treated as a finished step which needs no discussion beyond the mechanics, working with the generic platform suggested that this was not entirely true.

The text focuses on exploring the unwrapping method as it is the most popular. Three small changes are implemented to evaluate their impact on recognition performance. First, the effect of changes to the method of infilling noise areas within the unwrapped iris is evaluated. Then, the effect of noise removal using standard deviation is investigated followed by an evaluation of the use of oversampling with down-sampling using bicubic interpolation to resize the unwrapped iris to the desired template size.

The test results in this section do not include any times since normalisation times as measured by the generic platform are typically of the order of 0.01 seconds. Little variation

was seen during testing, so the times were removed to allow the focus to be on the recognition accuracy.

5.6.1 Intelligent Noise Infilling of Noise Within the unwrapped iris

In Masek's normalisation routine the areas of noise such as eyelids and eyelashes are marked on the unwrapped iris using NaN values. Once these NaN's have been transferred to the iris mask they must be replaced with real numbers in order to minimise their impact on the encoding process.

Masek's algorithm takes a copy of the unwrapped iris, fills the marked areas with the midrange value, finds the average over the modified image and then fills the marked noise areas on the original image with the updated average.

The object of these tests was to evaluate if and by how much recognition performance could be affected by this very small change during the normalisation stage. A modified version of the Normalise One code was used for this testing.

Instead of a blanket average, the first method tried uses intelligent infilling. This localises the averaging by taking the average of the non-NaN pixels around the each NaN location and finds the average, replacing the selected NaN with that average.

The second method was taken from the OSIRIS source code. Although there was insufficient time to convert all the OSIRIS code, the `fillWhiteHoles` algorithm was finished and was working correctly. OSIRIS uses this to remove specular reflections from the eye image before segmentation, but it was thought that it could also be used effectively to get the infilling values during normalisation.

The `fillWhiteHoles` algorithm repeatedly applies a dilation function to the whole image using convolution, until the starting image and convolved image show no change.

Table 5.9 – Recognition results from Segment Three, Normalise One with the intelligent infilling algorithm, Match One and Masek’s encoding routine

In-filling Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	99.7%	91.0%	4.3	1.3%
CASIA V2D1	99.9%	53.2%	3.0	4.9%
CASIA V2D2	92.7%	50.9%	2.2	13.7%
CASIA V3i	99.1%	93.0%	5.0	0.6%
IITD	97.1%	0.0%	5.1	2.1%
MMU V1	94.0%	1.6%	2.4	14.9%
MMU V2	92.0%	8.5%	2.2	13.1%
WVU Free	90.0%	20.8%	1.5	16.6%
Average	95.6%	39.9%	3.2	8.4%
Variance	9.9%	93.0%	3.6	16.0%

The recognition results in Table 5.9 show improvements with respect to the original results for Segment Three, see Section 5.3. However, CASIA V1 and WVU Free both show reductions in recognition performance.

Table 5.10 – Recognition results from Segment Three, Normalise One with the OSIRIS infilling algorithm, Match One and Masek’s encoding routine

OSIRIS in-filling Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	99.3%	79.5%	4.0	2.3%
CASIA V2D1	99.8%	61.9%	3.0	4.6%
CASIA V2D2	92.7%	36.9%	2.2	13.9%
CASIA V3i	98.9%	75.5%	4.8	1.0%
IITD	97.1%	0.0%	5.0	2.3%
MMU V1	93.8%	1.9%	2.3	15.9%
MMU V2	90.2%	9.4%	2.1	13.8%
WVU Free	86.3%	29.2%	1.5	18.3%
Average	94.8%	36.8%	3.1	9.0%
Variance	13.5%	79.5%	3.5	17.3%

The results in Table 5.10 also show the CASIA V1, IITD, MMU V2 and WVU datasets all have lower true match results. The CASIA V2D1, CASIA V2D2, CASIA V3i and MMU V1 all improve their true match results as compared to the original Segment Three algorithm. Average recognition performance is 0.4% less though and the results from using in-filling were slightly better for every dataset.

5.6.2 Noise Removal Using Standard Deviation Thresholding

Masek used a fixed threshold in his algorithms, applied to the eye image, to prevent noise such as eyelashes and eyelids being included in the encoding and matching processes. In Section 4.2 this was shown to be a poor method when testing with multiple image datasets.

The idea of thresholding during the normalisation process in order to remove noise was shown to improve recognition results, but these improvements did not occur at consistent threshold intensities across the datasets.

Correct identification of the threshold intensity for an image dataset would resolve this, but be unrepeatable in real world situations, such as security checkpoints where the input biometric data is unknown at design time. Once the iris has been unwrapped, however, the unwrapped iris pixels should be within the specific intensity range of the iris, with noise such as eyelids and eyelashes falling outside this intensity range.

This is achieved by taking the standard deviation of the unwrapped iris for each image and then setting all intensity values above and below the average by a set number of standard deviations to NaN. The remainder of the normalisation process remains unchanged.

Values of one, two, three, four and five standard deviations were tested to evaluate the impact of applying them to the unwrapped iris. Testing was carried out on the three more difficult image datasets to evaluate viability.

Table 5.11 to Table 5.14 contain the standard deviation based noise removal results. The 'None' column in these four tables lists the results obtained previously using Segment Three, see Section 5.3 and the best average results are shaded.

Table 5.11 – True match results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.

True Match	Standard Deviation					
	1	2	3	4	5	None
MMU V1	93.80%	93.30%	94.20%	94.00%	94.00%	94.00%
MMU V2	91.90%	92.00%	90.80%	91.10%	91.10%	91.30%
WVU	80.00%	91.30%	93.80%	93.80%	93.80%	91.30%
Average	88.57%	92.20%	92.93%	92.97%	92.97%	92.20%

The average values in Table 5.11 suggest that an improvement in recognition results can be obtained by applying thresholds at standard deviations between three and five, but four and five are the best.

Table 5.12 – Separability results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.

Separability	Standard Deviation					
	1	2	3	4	5	None
MMU V1	1.60%	1.50%	3.30%	1.70%	1.60%	7.40%
MMU V2	9.00%	5.40%	19.00%	9.00%	9.00%	0.60%
WVU	20.80%	30.80%	40.80%	22.50%	20.80%	12.50%
Average	10.47%	12.57%	21.03%	11.07%	10.47%	6.83%

The separability results in Table 5.12 suggest that using any standard deviation threshold will significantly improve the separability of the recognition results for the MMU V2 and WVU datasets, but will reduce them for the MMU V1 dataset. The best gains are to be had by using a threshold of three.

Table 5.13 – Decidability results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.

Decidability	Standard Deviation					
	1	2	3	4	5	None
MMU V1	2.40	2.40	2.40	2.40	2.40	2.4
MMU V2	2.20	2.20	2.20	2.20	2.20	2.2
WVU	1.30	1.50	1.50	1.50	1.50	1.4
Average	1.97	2.03	2.03	2.03	2.03	2.00

The decidability results in Table 5.13 are inconclusive, in that any standard deviation threshold other than one or none would give the best decidability.

Table 5.14 – EER results from applying the standard deviation thresholds to the unwrapped irises attained from using Normalise One with Segment Three, Segmentation Mask One, Masek’s encode and Match One.

EER	Standard Deviation					
	1	2	3	4	5	None
MMU V1	14.61%	14.53%	15.08%	14.87%	14.87%	15.19%
MMU V2	13.50%	13.03%	13.42%	13.40%	13.40%	13.50%
WVU	22.55%	15.16%	14.29%	14.14%	14.14%	15.10%
Average	16.89%	14.24%	14.26%	14.14%	14.14%	14.60%

EER is the only metric in this section where lower is better. The EER results in Table 5.14 indicate that a standard deviation threshold of four or five would be the slightly better threshold.

The results for this method show that thresholding can be advantageous when using standard deviation as the method of deriving the threshold. Separability aside, the results indicate that a small improvement in recognition performance can be achieved using a

standard deviation threshold of four or five. This improvement applies to all average metrics.

A very large improvement in separability can be achieved by using a standard deviation threshold of Three. However, this is less clear-cut as there is only a small improvement in true match rate and EER, and the same increase in decidability that is seen when using thresholds of four or five. The variability in the results when using a standard deviation threshold of three suggests it may be less suitable across a wider range of datasets.

5.6.3 Oversampling to Improve Noise Immunity

Oversampling is a common technique that can improve the quality of digital data by reducing aliasing and noise. In audio systems oversampling means acquiring data at a greater rate than the Nyquist minimum rate of two times the highest frequency.

For audio, oversampling, followed by down-sampling often produces a signal with less aliasing noise, it was considered that this might also be true for the iris image. No evidence could be found in the literature of oversampling followed by down-sampling being used in this function.

As already stated in Section 2.4, when the iris is unwrapped, it is sampled at a consistent number of radial sample points but the inter-sample spacing is altered to maintain even sample coverage across the iris. For Masek's algorithm, there are 240 radial lines and 20 sample points for each line.

This method increased the number of radial lines to the number of points in the pupil circumference, or 240, whichever was greater. This ensured that the maximum number of radial lines was being sampled, without duplicate samples being taken. The number of sample points was altered to be the smallest radial distance between either the pupil and iris boundaries or 20, whichever was smaller.

After sampling the iris image is down-sampled back to 240×20, using a bicubic resize, and processed as before.

Table 5.15 – Recognition results from Normalise One using oversampling with the developed iris boundary detection, Segmentation Mask One, Masek’s encode and Match One

Oversampling Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	99.9%	93.3%	4.90	0.9%
CASIA V2D1	99.7%	44.9%	3.10	4.8%
CASIA V2D2	92.3%	41.2%	2.20	14.0%
CASIA V3i	99.0%	95.1%	5.60	0.46%
IITD	97.3%	0.0%	5.70	2.0%
MMU V1	93.6%	1.6%	2.40	14.6%
MMU V2	91.2%	5.7%	2.20	14.1%
WVU Free	95.0%	44.2%	1.70	10.8%
Average	91.2%	40.8%	4.90	14.1%
Variance	95.0%	95.1%	3.10	10.8%

The results in Table 5.15 show small improvements in the IITD and CASIA V2D1 results over the original Segment Three results in Section 5.3, and a bigger improvement in the WVU Free results. This is counter-balanced by slight reductions in the MMU V1 and V2 results and no change in the remaining results. Overall, a slight increase in average recognition results from 95.5% to 96% can be seen.

5.6.4 Normalise Two

Each of the methods proposed in the previous three sub-sections made a very small improvement in recognition performance. Each method was stronger in some datasets than it was in others.

Looking at the results side-by-side it was noticed that, compared with the original results for Segment Three using Normalise One, at least one of the methods was either better or the same as the original. All of the methods improve the CASIA V1 results, two out of the three were better for CASIA V2D1 and V3i, IITD and WVU Free. The CASIA V2D2 and MMU V2 datasets had at least one result of the three that was better than the original.

This suggested that combining all three small improvements might lead to a bigger overall improvement. Based on the results, five was selected for the standard deviation threshold and then all three algorithms were added to Normalise One. The result was a small improvement across all the datasets except the MMU V1 and a large improvement in the WVU results. This new normalisation routine was called Normalise Two and for simplicity the combination of Segment Three with Normalise Two, Segmentation Mask One, Masek's encoding and Match One from Section 5.8 was named 'Flow One'. The results from Flow One are compared with those obtained using Segment Two and Three, OSIRIS, Masek and Li's segmentation algorithms using Normalise One in Figure 5.23.

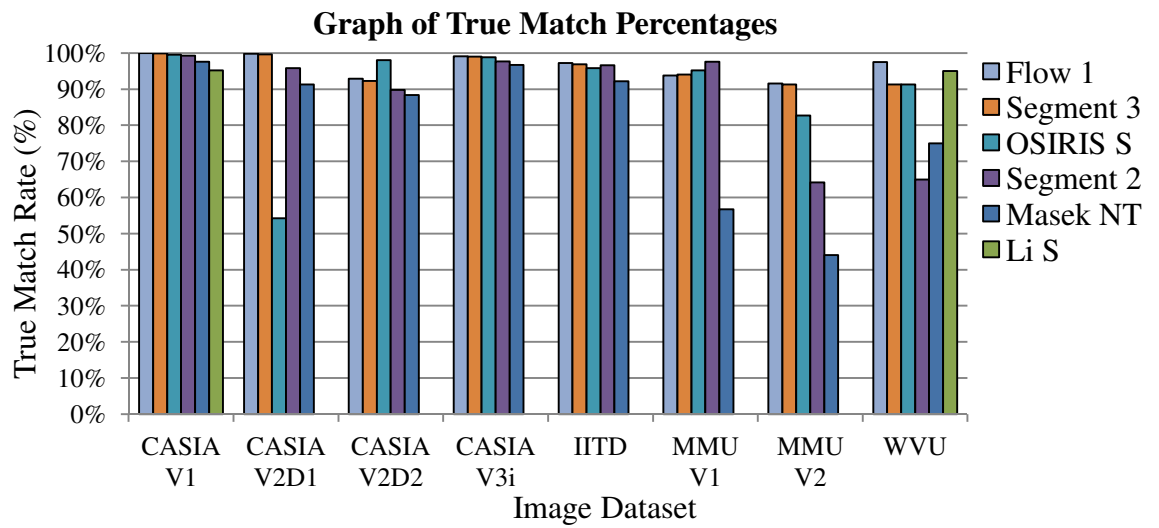


Figure 5.23 – The results from Flow One compared with those obtained using Segment Two and Three, OSIRIS, Masek and Li’s segmentation algorithms

Figure 5.23 shows that Flow One attains the highest results for six out of the eight datasets. The OSIRIS segmentation algorithm is the best for the CASIA V2D2 dataset and Segment Two is the best for the MMU V1 dataset.

Table 5.16 – Recognition results from Normalise Two using the combined normalisation improvements with Segment Three, Segmentation Mask One, Masek’s encode and Match One

Normalise Two Recognition Results				
Dataset	True Match	Separability	Decidability	Equal Error Rate (EER)
CASIA V1	100.0%	96.8%	4.9	0.9%
CASIA V2D1	99.8%	48.2%	3.2	4.4%
CASIA V2D2	92.9%	46.0%	2.2	13.5%
CASIA V3i	99.1%	96.4%	5.6	0.4%
IITD	97.2%	-	5.7	1.9%
MMU V1	93.8%	2.1%	2.4	15.2%
MMU V2	91.6%	2.1%	2.2	12.6%
WVU Free	97.5%	43.3%	1.8	10.8%
Average	96.5%	46.6%	3.5	7.5%
Variance	8.4%	94.7%	3.9	14.7%

The results in Table 5.16 show an average true match result of 96.5%. This is the highest result attained with any combination of algorithms. The cost of 0.2% reduction in MMU V1 true match rate is minimal when compared to the gains elsewhere in the results.

5.6.5 Conclusions

This section investigated three small changes to the normalisation function in an effort to see if improvements in recognition performance could be attained. The three methods, intelligent infilling, standard deviation thresholding and oversampling had variable results on their own, though the oversampling was strongest, but combined they returned the best results across the datasets.

Thus it was shown that small changes can have a measurable positive impact on recognition performance and that even stages which seem to have no possibility of improvement can still be altered to improve recognition performance.

The simplicity of carrying out these tests within the generic platform cannot be overstated. It would be interesting to see how other iris recognition algorithm results improve from using Normalise Two, especially Segment Two, as there was insufficient time to do this.

The next section discusses Segmentation Mask One

5.7 Segmentation Mask One (SM1)

Segmentation Mask One was written to replicate Masek's segmentation mask code without the intensity threshold and to extend it to cope with circular and elliptical eyelid boundaries.

As already stated in Section 4.1, Masek's algorithm applies a greyscale intensity threshold at greyscale level 100. His segmentation mask code can also only cope with eyelid boundaries that are horizontal lines.

With respect to the eyelids, some algorithms use lines [30, 29], some ignore them [27], others use other methods such as filtering and thresholding [30] and this research uses circles. Not all algorithms are 'fixed flow's, and so they need to use other people's functions to fill in the gaps during testing. In order to provide a fair test, the segmentation mask had to cope with as much variety as possible.

A simple case statement is used to check the type of each boundary (line, circle or ellipse) and then the appropriate algorithm is executed. For the top eyelid, all pixels within the image from the boundary upwards for the width of the iris are marked as NaN. For the bottom eyelid it is all the pixels from the boundary downwards for the width of the iris that are marked as NaN.

Once it was coded, SM1 was tested with Masek's code on all the selected test datasets. Using SM1 with Masek's flow, the results produced by the generic platform outputs showed no measurable difference between the results obtained using SM1 and the results from Table 4.7 in Section 4.1 using Masek's segmentation mask with the thresholding removed.

The next section discusses a very fast matching algorithm called Match One

5.8 Match One

A great many iris recognition algorithms use the Hamming distance to measure the differences between templates, [60][76][38][3][30][31][64] to list a few. In many ways, it has become the de facto standard within the iris recognition field. The Hamming distance comes from information theory and is typically used as a method of measuring the number of positions in which strings of binary data differ.

When carrying out initial testing using the generic platform, the analysis process was found to be very slow, taking hours, or days to complete. A significant proportion of this time was taken up by the matching routine. The main aim of Match One, therefore, was to reduce the time taken for testing.

Masek's Hamming distance algorithm was used as the default matching function for any algorithm that did not include its own matching code. Given that Masek's algorithm is quite slow, and that large amounts of testing were to be carried out, there was concern about how long this process would take.

Table 5.17 – Time to analyse each dataset using Masek’s Hamming match (hours:minutes)

Masek Times	
Dataset	Analysis Time (h:mm)
CASIA V1	1:10
CASIA V2D1	2:52
CASIA V2D2	2:36
CASIA V3i	13:38
IITD	11:33
MMU V1	0:23
MMU V2	1:53
WVU Free	00:00:42
Total	34:05:42

The ‘Total’ column in Table 5.17 shows the total hours taken to analyse each of the databases. An improvement in speed of 20% was set as the target for the Match One method, as even this modest saving would free up over four hours of test time over the eight datasets.

Masek’s match routine uses the Hamming distance technique originally proposed by Daugman for use with iris recognition. This allows strings of binary digits to be compared as, in normal iris recognition algorithms, the initial greyscale iris image is converted to a grid of binary digits by the encoding process.

The process of using the Hamming distance with and without masks is described in detail in Section 2.6. Masek’s code makes two notable changes to Daugman’s equation (2.6), the first is the use of variable code size, instead of Daugman’s fixed 2048-bit code and mask. Defining variable $N = \text{code } x \text{ size} \times \text{code } y \text{ size}$, allows for much more flexibility in the use of the Hamming distance algorithm, allowing it to function with rectangular iris codes of all sizes. However, any iris codes being compared must be the same size.

The other significant change is that Masek's code uses masks that are inverted with respect to the masks Daugman uses, so logical ones mark noise, zeros mark data. This marginally simplifies his encoding algorithm at the expense of more processing during the Hamming distance calculation. Equation 5.6 shows the modifications to equation 2.5 for these changes.

$$HD = \frac{1}{N} \sum_{j=1}^N (CodeA_j \oplus CodeB_j) \cap \overline{MaskA_j \cup MaskB_j} \quad 5.6$$

where, N is the number of bits in each mask and code, $CodeA_j$ is the first iris code and $CodeB_j$ is the second iris code, $MaskA_j$ is the first iris mask and $MaskB_j$ is the second iris mask, \oplus indicates logical XOR, \cup indicates logical OR and \cap indicates logical AND.

Using equation 5.6, the two masks, $MaskA_j$ and $MaskB_j$, are combined into one mask using a logical NOR function so that only areas that are valid in *both* masks are used in the comparison process. Once the iris codes have been combined into a single code using the logical XOR function the combined mask is logical AND'ed with it, thus removing any parts that should not be used for comparison. As discussed in Section 2.6, this can improve the accuracy of the Hamming distance that is returned.

Before any of these logical operations can be performed, one mask and its associated iris code must be rotated to compensate for rotations of the eye. This is a very important step and can significantly affect algorithm accuracy [28]. This rotation is only performed on one code and mask.

Therefore, using pseudo code, if the variable `iShift` is the number of shifts, the full process is:

```
For iLoop = -(iShift/2) to (iShift/2)
```

```
  Shift Code and mask 'B'
```

```
  combined code = Code 'A' XOR shifted Code 'B'
```

```
  combined mask = Mask 'A' OR Mask 'B'
```

```
  masked iris code = combined code AND combined Mask
```

```
  valid bits = the number of logical 1's in masked iris code
```

```

N = Code 'A' X * Code 'A' Y
Hamming distance = valid bits / N
endfor

```

Masek does all of this and so Match One had to do this as well. When evaluating Masek's code the slowest parts were the shifts. N is constant, so could be calculated just once outside the loop. In Masek's open-source code, his shifting algorithm shifts both the original code and mask by the required number of bits on each iteration of the loop (Figure 5.24). This is a very slow, memory bandwidth intensive method.

1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8
2	2	3	3	4	4	5	5	6	6	7	7	8	8	1	1
3	3	4	4	5	5	6	6	7	7	8	8	1	1	2	2

Figure 5.24 - A vector rotated using Masek's method

For Match One, a pointer into an extended array was used. This is a well-known solution to the problem of data shifting, and yet is not used by any of the open-source solutions that were available to this research at the time Match One was written. Only the C++ based OSIRIS uses this technique and the source code for that was not available until April 2012, over a year after Match One was developed.

The original code and mask are extended by grafting the copies of the last few and first few columns to the beginning and end respectively. Taking the first vector shown in Figure 5.25, the black box indicates a sliding window, which uses a set of pointers to the desired part of the vector.

7	7	8	8	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	1	1	2	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Figure 5.25 - An extended vector using Match One, with sliding window indicated

These pointers can be moved very easily, with very little mathematics or memory operation requirements. In MATLAB this is very simple, as a new variable can be assigned

to the selected vector elements and so long as the contents of the vector do not change, the new variable will just point to the part of the original that is applicable.

When writing code in MATLAB, the pointer must be increased by two for each iteration. This is because MATLAB stores each binary digit as a character with each character in its own array index. Each value in the iris code can be a value between zero and three, requiring two binary bits, so two array members are needed per value. Hence, the repeated numbers on Figure 5.24 and Figure 5.25.

If written in a lower level language such as C++ where it is practical to perform the match in code rather than use a library, the same sliding window would likely be achieved by just wrapping the pointers around. No extra copying or memory would be required, therefore, when using pointers in C++, it would also be possible to store four, two-bit, values inside one byte. Unfortunately, these options are not available in MATLAB.

This simple change combined with moving the N calculation out of the main loop had a massive impact on the required processing time. Using De Morgan's theorem, $\overline{MaskA_j \cup MaskB_j}$ becomes $\overline{MaskA_j} \cap \overline{MaskB_j}$, (Equation 5.7) thus, the mask negation as well as another subtraction could also be moved out of the loop.

$$HD = \frac{1}{N} \sum_{j=1}^N (CodeA_j \oplus CodeB_j) \cap \overline{MaskA_j} \cap \overline{MaskB_j} \quad 5.7$$

where, N is the number of bits in each mask and code, $CodeA_j$ is the first iris code and $CodeB_j$ is the second iris code, $MaskA_j$ is the first iris mask and $MaskB_j$ is the second iris mask, \oplus indicates logical XOR and \cap indicates logical AND.

De Morgan's theorem is one of the fundamental laws of logic, it states that $\overline{\overline{A} \cup \overline{B}} = \overline{\overline{A}} \cap \overline{\overline{B}}$ and that $\overline{\overline{A} \cap \overline{B}} = \overline{\overline{A}} \cup \overline{\overline{B}}$. This means that the change to the Hamming distance equation is in the manner of a substitution, as even though the equation has transformed, the answer is exactly the same for the same values of $Mask A_j$ and $Mask B_j$.

It would have been more logical to have simply created the mask inverted in the first place, however, this routine was being designed to fit in with Masek's flow. Masek's

normalisation routine creates the mask this way and his encoding routine expects to see a mask this way round.

The analysis times using just this shifting enhancement decreased, on average, by slightly more than 50%, but it was still taking over 16 hours to complete testing on all eight of the datasets. Therefore, having addressed the shifting performance, other areas of performance enhancement were investigated.

One area that showed promise was the data format used. In MATLAB binary numbers are stored in a data-type called 'logical' and each 'logical' variable can be either one or zero. This means to build up a two digit binary number, a two element 'logical' array is needed and these elements are stored in separate bytes in memory, therefore, for each two-bit phase-quadrature encoding, two array elements are required.

This storage format meant that the code and mask were double the length that they would have been in a lower level language where the bits would be stored consecutively within memory locations. Storing consecutively was quickly discounted as it would again require physical shifts and would thus be slow.

As already described, the phase-quadrature encoding used two bits per pixel within the original normalised iris image to encode the iris texture. The mask also used two bits per pixel in the original normalised iris image, but the bit pairs in the mask were always the same value as each other, so '11' or '00', never '10' or '01'. This was necessary as the mask marked which pixels within the original normalised iris image and there are two bits for each pixel in the code that must be masked.

By placing each of the bit pairs within the iris code into individual integers, it was possible to reduce the mask to a single digit per original pixel. This conversion process had the intended effect of halving the length of the array, thus halving the number of elements and the number of matrix operations per step.

Using mathematical subtraction was 50% faster, but produced results that showed a reduced recognition performance. Evaluating the results of the subtraction, versus the results of the binary XOR operation used in the Hamming distance showed that they were very similar. Table 5.18 shows these two results side by side.

Table 5.18 Comparison of the difference calculations versus the HD. A and B are the input data $|A - B|$ is the magnitude of the difference calculation, $A \oplus B$ is the logical XOR of A and B in binary, and the last column shows the HD bit count result from the $A \oplus B$ column.

A	B	$ A - B $	$A \oplus B$	$A \oplus B$ HD Count
0	0	0	00_2	0
0	1	1	01_2	1
0	2	2	11_2	2
0	3	3	10_2	1
1	0	1	01_2	1
1	1	0	00_2	0
1	2	1	10_2	1
1	3	2	11_2	2
2	0	2	11_2	2
2	1	1	10_2	1
2	2	0	00_2	0
2	3	1	01_2	1
3	0	3	10_2	1
3	1	2	11_2	2
3	2	1	01_2	1
3	3	0	00_2	0

The 'A' and 'B' columns in Table 5.18 list the input values, listed in Gray code order, with the ' $|A - B|$ ' column listing the magnitude of A minus B. The third column, ' $A \oplus B$ ' lists the result of the logical A XOR B operation displayed in binary, with the final column showing the number of logical ones in the ' $A \oplus B$ ' column. The ' $|A - B|$ ' column in Table 5.18 is the result returned by the difference calculation, the ' $A \oplus B$ HD Count' column is what was needed. The 3's in the ' $|A - B|$ ' column were consistently erroneous, so could be replaced using a fast matrix search and replace.

5.8.1 Results

Table 5.19 – Analysis times using Masek’s Hamming match versus Match One

Analysis Times (h:mm)			
Dataset	Masek	Match One	Reduction (%)
CASIA V1	1:10	0:19	72.4
CASIA V2D1	2:52	0:48	71.5
CASIA V2D2	2:36	0:51	67.3
CASIA V3i	13:38	5:30	59.7
IITD	11:33	2:36	77.5
MMU V1	0:23	0:06	73.2
MMU V2	1:53	0:33	71.4
WVU Free	00:00:42	0:00:13	69.0
Total	34:05:42	10:44:21	
Average			70.3

The Recognition results from this test output by the generic platform were indistinguishable from those achieved with Masek’s matching code, however, Table 5.19 shows an average of 70.3% reduction in the time taken for Match One when compared to Masek’s code. This decrease is more starkly demonstrated by the total times of 34 hours for Masek’s matching algorithm against the 10 hours 44 minutes for Match One, a saving of 23.3 hours. Using Match One would allow three sets of experiment results to be analysed in the time Masek’s algorithm needed for just one analysis.

Figure 5.26 shows the results in Table 5.19 divided by the number of comparisons made on each dataset. This showed the amount of time taken to calculate the Hamming distance between two iris codes for a given image dataset and algorithm.

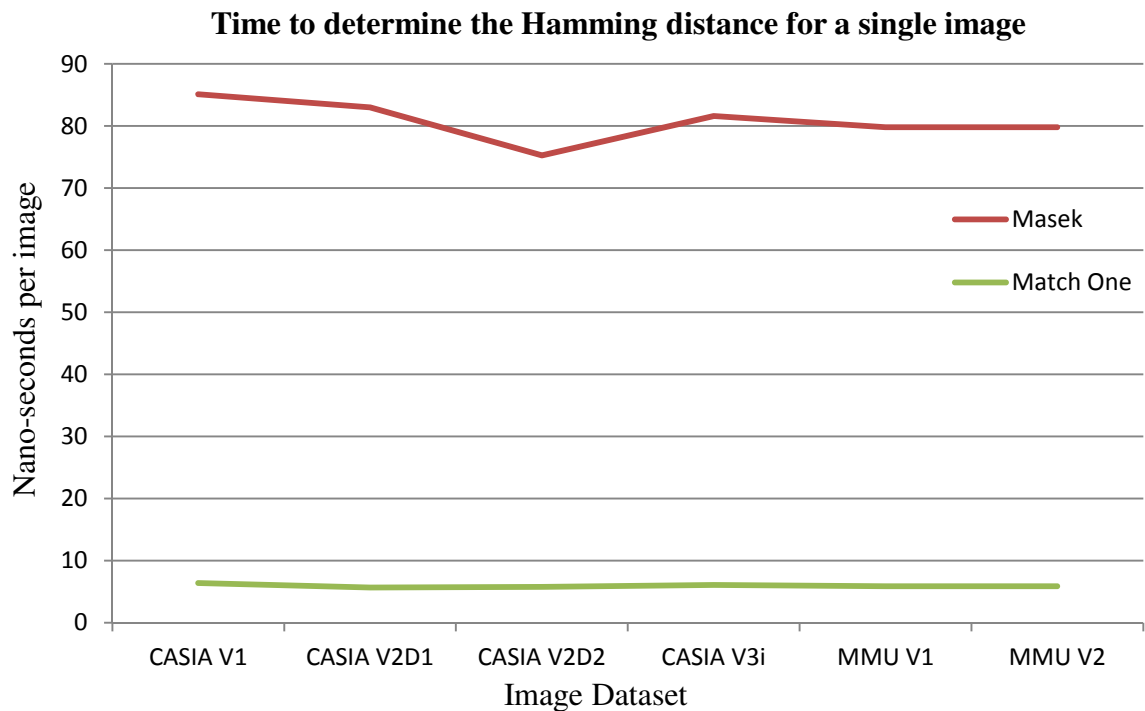


Figure 5.26 – Average time to find the Hamming distance between two iris codes

The graph in Figure 5.26 clearly shows the performance increases of changing from Masek's code to Match One. The variation across the datasets can be attributed to system load at the time the test was run.

5.8.2 Conclusions

This section looked at the development of a Hamming distance algorithm that had the sole purpose of decreasing the time taken to analyse iris code recognition performance. The aim was for a 20% reduction in analysis time and no change in the recognition performance. Table 5.19 shows that this was substantially bettered with an average reduction in analysis time across the datasets tested of 70.3% when Match One is used.

The speed of this algorithm was remarkable in the generic platform environment, especially since no effect of doing this could be measured on any of the dataset results when compared with Masek's Hamming distance code (see Table 4.7).

Testing that would previously have been carried out using Masek's Hamming distance code, hereafter used Match One, unless specifically stated otherwise.

The next section discusses a small GUI designed to enable the recognition results in results databases to be compared

5.9 Compare Result Databases

The errors in Li's code meant that there was a need to be able to compare result databases from the 'fixed flow' and 'custom flow' of each algorithm. In essence, the same algorithm should return the same recognition results, though the timing might be expected to vary a little. For Masek and Antonino's code this was the case, however, for Li's code it was not.

Again, the nature of Li's code, and in particular the structure and formatting of the code, made it very easy to make simple mistakes when extracting the two flow types. Comparing the results databases from each flow type would allow the problem areas to be pinpointed quickly.

To this end, a small GUI was created whose sole purpose was to compare the results from the segmentation, normalisation and encoding outputs. Figure 5.27 shows the GUI with the major elements marked with large blue numbers. List (1) and (2) both contain the full list of result databases available. The user selects the source database from list one and the database to compare with from list two

The check box at (3) allows only the first ten records to be compared, reducing the output data. Pressing the button labelled 'Go' at (4) sets the comparison test going.

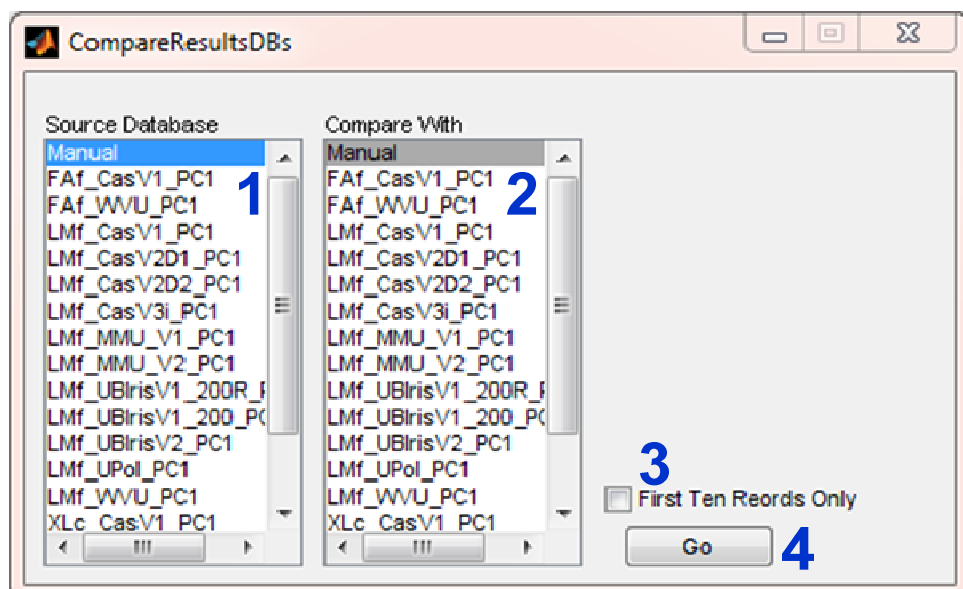


Figure 5.27 – Compare Result Databases GUI

Results are returned in a test file in the source database directory. The text file has a standard name of Compare-EncodeDB_<YYYY-MM-DD>.txt, where <YYYY-MM-DD> is replaced with the test date in that format. This is an example of the output from this routine:

```
Output of CompareResultsDB.m
Written by Chris Ponder, 2011-04
Encode Comparison started: 2011-10-20 12:02
-----
Source DB: XLc_CasV1_PC1   Compare DB: XLf_CasV1_PC1

Record: 1
-----
Source Image: 001_1_1Compare Image: 001_1_1
Iris.ShapeType fields are both ellipses
Iris.Shape fields match

Pupil.ShapeType fields are both ellipses
Pupil.Shape fields match

TopLid.ShapeType fields are both no type
TopLid.Shape fields are empty

BotLid.ShapeType fields are both no type
BotLid.Shape fields are empty

Code fields do not match
Mask fields do not match

<...>

-----
Total Differences: 1505
-----
```

This shows the name of the script, the author and the type of output at the top. The source and comparison databases are then listed followed by the list of record comparison results, in this case just the first of 756 comparisons. The “< . . .>” indicates the remaining 755 records, and then there is a summary line indicating the number of differences, 1,505 for this test.

This GUI proved to be a huge help in determining where the problems had arisen between the ‘fixed flow’ and the ‘custom flow’ in Li’s code. This allowed the differences between the ‘fixed flow’ and the ‘custom flow’ to be corrected so that the generic platform output results for both that showed no measurable difference.

CompareResultsDB is a very quick script to execute and enhances the debugging ability within the platform significantly. This small GUI also enhances the development cycle, allowing the output from different versions of the same script to be examined for changes.

The next section compares the generic algorithms and methods presented in this chapter with the available literature of the time

5.10 Comparison of the Methods Developed with the Literature

Eight papers from the literature were investigated in detail as part of this research. Literature from Lin *et al.* [30], Lu [60] and Shamsi *et al.* [62] was discussed, implemented and evaluated in Section 4.5. The five remaining papers were from Aydi *et al.* [87], Emerich *et al.* [88], Gu *et al.* [86], Hollingsworth *et al.* [89] and Murugan and Savithiri [33]; these are discussed in this section.

Aydi *et al.* [87] proposed some small modifications to Masek's open-source segmentation algorithm to reduce processing time and increase accuracy. Aydi *et al.* change Masek's image resize from bicubic to nearest neighbour, add fixed gamma correction (of an unspecified amount), apply intensity stretching, and replace the Canny edge detection used by Masek with an unnamed matrix used to compute the gradients within the image.

The results presented by Aydi *et al.* show that testing with the CASIA V3i dataset showed a 4% increase in segmentation accuracy and 68% reduction in code creation time when compared to Masek's algorithm – a clear improvement over Masek's algorithm. However, using Segment One with Normalise One increases recognition performance by 20% and reduces code creation time by 95% when compared with Masek's algorithm.

Aydi *et al.* highlight the same message as Normalise Two, that small changes can bring about measurable improvements in recognition accuracy. However, the pupil and iris accuracy results are clearly decided by manual evaluation and there exists no automated way of evaluating this except the method discussed in Section 5.4, which was not available to Aydi *et al.*.

The manual evaluation of segmentation results makes it difficult to reliably compare this method to other works as the results are subjective. The alteration to Masek's image resizing method from bicubic to nearest neighbour was implemented and tested on the CASIA V1 dataset. The results showed improvements in the true match rate (97.6% to 98%) and separability (21.1% to 70.9%) with no change in the decidability or EER. This is only one out of the three changes proposed by Aydi *et al.*. There was insufficient time for additional investigation.

Further work could look at recreating the algorithm, though without more details this would require significant experimentation to obtain some of the values such as the fixed gamma correction. This would be an ideal scenario for the generic platform as repeated testing of algorithms using different parameters is a task at which this GUI excels.

Gu *et al.* [86] present a segmentation method which is directly compared with Masek's. They pre-process the original iris image by removing specular reflections and performing image smoothing. Intensity values of 200 and above are assumed to be specular reflections and replaced with values based on the surrounding pixels. The image is then smoothed using an unspecified smoothing operation.

The iris-pupil boundary is detected by selecting 10 of the lowest intensity pixels, and then standard deviation is used to find the boundary of the pupil-iris, using the assumption that points within the pupil will have a lower standard deviation than the points around the edge of the pupil.

The outer iris boundary by measuring the gradient between a minimum and maximum boundary range, with the highest gradient selected as the iris-sclera boundary.

Gu *et al.* use the CASIA V1 dataset to test their approach, although they don't discuss or address the flaws in the CASIA V1 dataset as described earlier by Phillips [137]. It would be interesting to see how the algorithm proposed would perform with modern iris datasets. Manual evaluation of segmentation results is again used, this time presenting a 41.7% improvement over the results obtained using Masek's open-source algorithm. The algorithm from Gu *et al.* is stated to be 52% faster than Masek's.

The standard deviation method as presented is strongly dependant on the pupil modifications present in the CASIA V1 dataset and would almost certainly be less successful on images that were not modified. Unmodified images often contain significant variations in intensity in the pupil region that would render this standard deviation based methodology useless.

The use of manual evaluation of the results is yet again suspect and incomparable with the generic platform output. Further work could look at recreating the algorithm using the

generic platform, but, as described in the literature, there are no compelling reasons to investigate this method further.

Murugan and Savithiri [33] present experimental results using Masek's iris recognition algorithm, that evaluate the potential to maximise recognition results by restricting the parts of the iris used for the matching process. Five portions are compared, the whole iris, the inner half of the iris ring, the outer half of the iris ring, the top half of the iris and the bottom half of the iris. Murugan and Savithiri use the MMU dataset for testing. Previous literature has found that the inner portion of the iris ring contains the most useful information [153, 154, 155, 156], though Hollingsworth *et al.* [99] suggested that the central one-third band could actually be better.

Murugan and Savithiri's results showed that removing the top half or outer ring of the iris had no impact on recognition results, but removing the bottom half or inner band reduced recognition performance. These results corroborate previous literature [153, 154, 155, 156] as well as preliminary tests carried out using the generic platform to investigate the literature from Hollingsworth *et al.* [99].

The use of the generic platform to further investigate the claims by Murugan would be a simple and worthwhile further work.

Emerich *et al.* [88] present a novel encoding method that uses TESPAN DZ matrices instead of Daugman's phase quadrature encoding, both of which are described in Section 2.5. They also evaluate seven wavelet transform algorithms to see which provides the most effective results. Masek's algorithm is used for the segmentation, normalisation and matching stages of the iris recognition process.

Unusually, Emerich *et al.* use the UPOL image dataset, which is a very low noise, high-resolution dataset, as discussed in Section 3.7. The best result achieved by Emerich *et al.* was 98.92% true match rate, which when compared with the results of between 92.9% and 100% obtained using Segment Three with Normalise Two is a comparable result.

The results obtained by Emerich *et al.* are interesting and would bear further investigation using the generic platform. It is a shame that a baseline result using Daugman's phase quadrature encoding is not presented for comparison, though this is something that the

generic platform could easily be used to produce. Without this baseline, it is difficult to know if the TESPARDZ encoding improves recognition accuracy.

Hollingsworth *et al.* [89] present a matching algorithm that uses the location of fragile bits in an iris code to improve match results. As described in Section 2.6, fragile bits are bits that change value across iris codes created from different images of the same iris, so represent changeable noise within the image. Previously Hollingsworth *et al.* proposed masking fragile bits to improve accuracy [99].

Hollingsworth *et al.* used a C/C++ reimplementation of Masek's algorithms called IrisBEE with the CASIA V3i image dataset. As with Normalise Two and Aydi *et al.* [87], Hollingsworth *et al.* show that a small change to part of the process can lead to a measurable improvement in recognition results. The 8% improvement in recognition rates over the baseline algorithm are not as strong as the 20% improvement obtained using Segment One with Normalise One, but they are still very worthwhile and in an area of the iris recognition process that is typically neglected.

The only difficulty with this method is the use of pre-knowledge of an eye to determine the fragile bits. When used with the image datasets available this is likely reliable, as many of these datasets have been seen to be very consistent between images of the same eye, as discussed in Appendix Appendix B. In the real world, with significant time between image capture sessions it is doubtful that the location of these noisy bits would be consistent.

None of the literature discussed in this section is a generic platform capable of accelerating the development and evaluation of full or partial iris recognition algorithms and none of them suggest any improvements to the normalisation process.

The literature from Aydi *et al.*, Gu *et al.* and Murugan and Savithiri all address the segmentation stage of the iris recognition process. The method from Gu *et al.* is significantly flawed and dependant on the modifications within the CASIA V1 dataset, however, Aydi *et al.* and Murugan present methods that should be investigated in further work. Though their results do not improve the recognition results by as much as Segment Three with Normalise Two, they propose some interesting ideas.

The literature from Aydi *et al.*, Hollingsworth *et al.* And Murugan and Savithiri all support the idea that small changes to one part of the iris recognition process can measurably improve the recognition rates, as was shown for Normalise Two.

It would also be of interest to investigate the use of TESPARDZ encoding and fragile bits further. Again this is a task that the generic platform would support very well.

The overall chapter conclusions are discussed next

5.11 Chapter Conclusions

Seven algorithms, an automated segmentation-boundary evaluation method and a result database comparison GUI were presented in this chapter and the algorithms and methods developed were compared to the literature.

Three segmentation algorithms were presented, with the second and third methods proposing novel methods of isolating the boundaries of the eye. An automated segmentation evaluation method and the results were then discussed.

Two normalisation algorithms were introduced, one that added the ability to normalise using elliptical boundaries, the second was an improved normalisation routine that used a combination of small improvements to produce a bigger overall improvement. An improved segmentation mask algorithm with the ability to handle ellipses and no thresholding was also introduced.

The final method was a much faster matching algorithm that was designed to execute very quickly within MATLAB, this reduced algorithm evaluation times using the generic platform, enabling iris algorithm evaluations to be carried out much more quickly.

Segment One gave poor performance as a recognition algorithm, but the results were useful in shaping the form of Segment Two.

Segment Two was much more effective at determining the iris boundaries within images of eyes, and this was shown in the recognition performance. Segment Two was also very fast at performing the segmentation due to the reduction in edge image points and the use of the developed sum-of-columns method to find the iris-sclera boundary.

Segment Three was the most effective algorithm, outperforming all the open-source code that was available at the time of development as well as all the algorithms developed for this research. It was also a match for the speed of Segment Two, without the need to resort to thresholding.

The automated segmentation code showed great potential, with results that corroborated manual evaluation of segmentation boundaries.

Normalise One added the ability to normalise iris images when the segmentation boundaries were ellipses, as well as circles. This enabled the evaluation of the segmentation algorithms by OSIRIS and Li.

Normalise Two improved recognition rates across all metrics for no measurable performance impact. The automatic segmentation evaluation code showed useful results which will improve and become more scientifically acceptable as more image boundaries are graded by more individuals.

Segmentation Mask One added the ability to use elliptical eyelid boundaries for the creation of the segmentation mask. It also removed the thresholding operation included in Masek's segmentation mask algorithm.

The matching algorithm Match One exceeded all expectations and reduced the analysis time for datasets within the generic platform to a truly usable level.

Use of the generic platform allowed rapid development and testing of the segmentation algorithms, giving feedback in just a few hours as to the efficacy of the selected algorithm.

The combination of Segment Three, Normalise Two, Segmentation Mask One and Match One with Masek's encoding algorithm into a new flow called Flow One produced very effective results across all eight datasets. This was the widest range of testing seen across all the literature and was shown to be the most effective.

None of the literature was found to be a generic platform capable of accelerating the development and evaluation of full or partial iris recognition algorithms and none suggested any improvements to the normalisation process.

Aydi *et al.*, Murugan, Emerich *et al.* and Hollingsworth *et al.* present methods that should be investigated in further work and they all support the idea presented by Normalise Two, that small changes to one part of the iris recognition process can measurably improve the recognition rates. The generic platform would be ideal for the task of further evaluating the methods discussed.

The thesis conclusions and proposed further work are presented next

6 Conclusions and Future Work

6.1 Summary and Conclusions

The goals of this thesis were to create a generic platform [112] tool to aid in the development of iris recognition algorithms and to use that platform to research and develop novel iris-recognition algorithms. The overall objectives of the work in this thesis were:

- to produce a novel, flexible, easy to use, easily extendable software platform that enables iris recognition developers to experiment with a variety of algorithms on real data (Chapter 3)
- to use the platform to develop novel, efficient iris recognition algorithms and compare these to the current state-of-the-art (Chapter 5)
- to compare the performance of algorithms across image datasets (Chapters 4 and 5)
- to create a system capable of improving the reliability and consistency of reported iris recognition results (Chapter 4)

A literature search for iris recognition algorithms found that nothing currently exists that provides the ability to develop and test iris recognition functions. Most are simply showcases for a particular algorithm. The generic platform presented in this thesis is very flexible, and is easy to use and extend.

The generic platform was used to successfully test existing algorithms from Masek [31], Antonino [38], JIRRM [40], OSIRIS [110, 39] and Li [27]. Methods proposed in papers by Lin *et al.* [30], Lu [60] and Shamsi *et al.* [62] were also implemented and evaluated within the platform. However, it was difficult to implement some of the algorithms found in the literature as in many cases insufficient information was given to allow the results to be recreated.

The evaluation of five open-source iris recognition algorithms returned very mixed results. Algorithms from Antonino and JIRRM demonstrated very poor iris recognition capability, whereas the algorithms from Masek, Li and OSIRIS were much more capable. OSIRIS gave especially good recognition results across all eight datasets.

Seven iris recognition algorithms were developed using the generic platform and discussed in Chapter 5.

Table 6.1 – Summary of average recognition results and average code creation time per image for all the algorithms tested

Results Summary						
Algorithm	True Match	Separability	Decidability	Equal Error Rate (EER)	Average Code Creation Time (s)	Thesis Section
Masek No Threshold	80.2%	18.3%	2.0	19.2%	12.9	4.2
Antonino	58.1%	5.0%	1.1	31.4%	12.8	4.3.1
Li	95.1%	56.7%	2.2	10.5%	1.3	4.3.2
JIRRM	60.9%	3.9%	0.9	35.0%	10.5	4.3.3
OSIRIS	89.4%	29.9%	2.2	13.6%	0.2	4.3.4
Lin	59.0%	4.3%	1.0	30.4%	0.1	4.5.1
Lu	61.4%	3.0%	1.0	30.0%	11.1	4.5.2
Shamsi-Lin	38.8%	1.0%	0.5	42.0%	0.4	4.5.3
Segment One	54.1%	2.6%	0.9	34.7%	20.8	5.1
Segment Two	88.3%	21.5%	2.3	13.1%	0.7	5.2
Segment Three	95.5%	18.5%	3.2	8.4%	0.8	5.3
Flow One	96.5%	46.6%	3.5	7.5%	0.8	5.6

Masek's results in Table 6.2 are highlighted as they were used as the benchmark during testing. Li's results are shaded because the averages shown only include results from the two image datasets that Li's algorithm completed processing on, thus they do not represent a fair comparison to the other results. The results from Lin, Lu and Shamsi-Lin are shaded because these algorithms did not successfully segment images so the results do not present a useful comparison of capability.

Each of the seven iris recognition algorithms introduced in Chapter 5 introduced novel iris recognition methods, but significant improvements were found using Segment Two and Segment Three. However, the best overall results came from combining the best algorithms from each stage together, which illustrates the point that good iris recognition is achieved by incrementally adding several steps to the process, with each improving matters slightly, until the overall result becomes acceptable. There does not appear to be a single “magic bullet” that will give 100% accurate recognition in one move.

Table 6.3 – Comparison of the total analysis time when using Match One as compared to using Masek or Antonino’s algorithms

Analysis Times Summary		
Algorithm	Analysis Time (h:mm)	Section
Masek Hamming	34.05	4.2
Antonino	25:10	4.3.1
Match One	10:44	5.8

Match One is a MATLAB-optimised matching routine that, when used with a fast segmentation algorithm like Segment Two or Three, or if pre-calculation is available, enabled the code creation and testing of all eight datasets used in the text to be completed within 12 hours. This compares favourably with Masek’s flow, which takes over three days to complete the combined encoding and matching processes for all eight datasets, of which 34 hours can be seen to be the analysis time.

All fourteen algorithms tested in Chapters 4 and 5 were tested against the widest selection of image datasets seen in the literature and presented a significant challenge for the algorithms under test.

Using the generic platform the same four graphs and text file summary, with the same detailed results, are output for every single algorithm tested. This ensures that any algorithm tested within the generic platform is tested to the same high standards, so improving the reliability and consistency of reported results.

Thus it can be seen that all four of the main overall objectives listed in Chapter 1 were met. However, these were overall aims for this research, the specific aims for the generic platform tool were:

- it had to include a development environment that promoted fast development and be usable on different operating systems. (Section 3.1)
- it had to be able to output results in a form that would provide consistent, measurable, feedback of algorithm performance, thus, facilitating improved understanding of iris recognition (Section 3.2)
- it had to be possible to load and / or add different iris recognition operations without re-writing or recompiling the platform. (Section 3.3)
- it had to have all the different operations required for iris recognition in-built, but the default operations had to be replaceable with user-defined operations (Section 3.4)
- it had to allow the user defined operations and the in-built operations of the same type to be interchanged quickly and easily to facilitate process evaluation (Section 3.5)
- it had to provide the ability to quickly and easily load multiple image datasets into the GUI (Sections 3.6 and 3.7)

The selection of MATLAB as the language and development environment ensured that the development cycle within the generic platform is fast, easy and truly cross platform. The MATLAB environment provides a very fast development cycle with no need to compile between code changes and a powerful debugger. The generic platform was tested under Linux and Windows 7 and operated correctly on both operating systems.

The addition of pre-calculation to the generic platform significantly reduced code creation time when the segmentation algorithm and image dataset had been used together previously.

Results obtained showed that there is no measurable time penalty for using the platform when compared to running the algorithms standalone. When compared to the very labour intensive manual methodology employed by Masek and others [62, 60], the platform is

much faster and its automated nature allows the user to continue with another task while the results are calculated.

The platform was demonstrated to output consistent and clear results that are directly comparable to the others produced by the platform. This allows very fast evaluation of the performance of an algorithm. The automatic segmentation evaluation proved remarkably useful as, even without manual grading of the segmentation boundaries, the results returned closely reflected the manual evaluations carried out. The “compare-results” GUI enables differences in results to be investigated in depth.

This consistent, measurable feedback ensures that the generic platform is fully capable of providing the necessary benchmarking and evaluation for the task of developing iris recognition algorithms.

The design of the platform allows new code modules to be added to the GUI, making them available for testing and evaluation without any recompilation of the platform itself.

All the open-source algorithms were converted to run within the generic platform in some form, thus providing in-built algorithms and modules to perform all the stages of iris recognition. Fast swapping of these functions is facilitated by the layout of the GUI which lists all the functions available to the user.

Using directory locations and naming conventions for the storage of image datasets, results databases and code modules, provides a very fast and simple method of managing the data used by the generic platform. The skills required to create directories and files using the host operating system will already be familiar to anyone capable of developing an iris recognition algorithm, so very little new must be learned.

Thus it can be seen that the detailed aims for the generic platform were also met in full. As the time consuming data management tasks are taken care of by the generic platform, the user can concentrate on the task of developing new recognition algorithms.

Overall, the generic platform is a fast and effective way of developing and testing recognition algorithms. Without the need to implement initialisation, debug, output, test and analysis code, the development focuses on the recognition stage or algorithm being

implemented. Adding datasets, flows and modules is also very straightforward, greatly simplifying the development of new iris recognition methods.

The results in Chapter 4 demonstrated that the platform was significantly faster and easier to extend (with additional algorithms and image datasets) than manually writing every aspect of the recognition process. Even the partial conversion of OSIRIS from C++ was only significantly slowed by the need to extract the code from its C++ classes and track down a MATLAB equivalent for the OpenCV functions used.

Although the project lacked direct industrial input, choosing to publish the generic platform as an open-source iris recognition development tool meant that the project was strongly focused in the same manner as it would have been had there been an industrial sponsor.

With the completion of this work the generic platform can now be released for general use. Two websites will be used, the MathWorks MATLAB Central web site, <http://www.mathworks.co.uk/matlabcentral/> and Source Forge, <http://sourceforge.net/>. The code will be released under an open-source licence with the short 'Quick-Start' user guide in Appendix Appendix A and a more comprehensive technical user guide that will be implemented as an HTML wiki on SourceForge. The platform will then to be advertised via email to researchers and companies in the field of iris recognition.

Because of lack of time there are a number of areas that were not investigated and features that were not implemented. No testing was carried out under Mac OS, splines are only supported in the data-structures, not in the code, and more encoding algorithms would have been useful. The following section discusses some future work that could be carried out.

6.2 Future work

This section discusses potential future work for the generic platform and iris recognition algorithms that were evaluated and created.

The impact of fixed intensity stretching and gamma correction on segmentation performance with the selected image datasets was surprising. Figure 6.1a shows image 010204 from the MMU V2 image dataset, segmented by Segment Three with a fixed gamma correction of 2.2 applied before edge detection. Figure 6.1b shows the same image segmented by the same segmentation process but without any gamma correction.

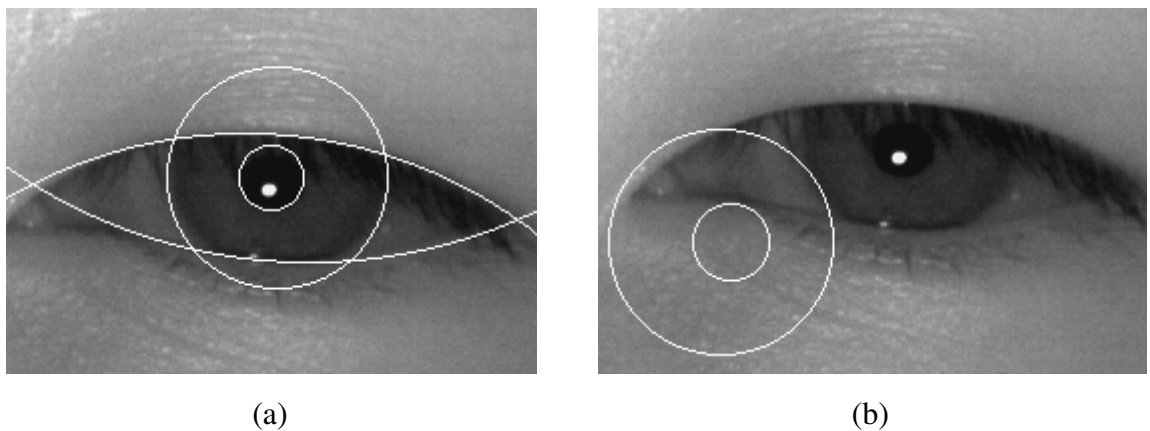


Figure 6.1 – Image 010204 from the MMU V2 image dataset, segmented by Segment Three (a) with a fixed gamma correction of 2.2 applied before edge detection, (b) without any gamma correction.

A similar effect was seen across image datasets, yet very little discussion was found in the literature on the subject of improving recognition rates using intensity stretching strategies. Work was carried out on fixed intensity stretching and fixed gamma and included in Segment Two and Three.

Automatically adjusting the contrast, together with gamma correction, might improve the consistency of recognition performance across different image datasets, and this should be investigated further. Chochia [150], Lee *et al.* [72] and Doustar and Hassanpour [73] have proposed adaptive gamma correction methods, and Grundland and Dodgson [157] proposed a polynomial-based automatic contrast enhancement method. While these are general image processing proposals, they may be applicable to this investigation.

Some initial work was carried out on replacing with ellipses the circles used by Segment Three. An initial manual inspection indicated improved pupil boundaries, but recognition performance was reduced. Further manual evaluation is needed to determine the reasons for this.

The automatic segmentation evaluation returned results that were mostly consistent with manual evaluation, but further work is needed to grade the boundaries used for reference and to validate the results returned.

The generic platform will become even more useful and effective as more algorithms are added to it. Considerable time was spent looking at segmentation and normalisation, and matching using Daugman's method. Work was started on Birgale and Kokare's [83] methods and their ring based normalisation was successfully implemented, but, there was insufficient time to implement the encoding and matching routines necessary to perform testing. Inclusion of more algorithms such as this would improve the development experience within the generic platform.

Rathgeb *et al.* [102], Daugman [78] and Hollingsworth *et al.* [99] all suggest that shifting iris codes with respect to each other during the matching process is essential for achieving good recognition performance. An initial evaluation of matching rotations against recognition performance was carried out and found that less rotation could sometimes be better. Further investigation is needed to evaluate this.

It would also be useful to investigate the proposal by Hollingsworth *et al.* [99] that using the central ring of the iris instead of the full range for matching might improve recognition rates. Again, a preliminary investigation showed mixed results, but further work is needed to validate this.

For the generic platform flow type checking, allowing the platform to automatically prevent incompatible modules from being selected would be very useful. Comma separated value (CSV) output of results, FAR, FRR, Hamming distribution, etc., for manipulation in a spreadsheet would help with the comparative evaluation of results. It would improve readability if HTML output was provided, as this would allow text to be combined with graphs of results.

More result formats would make it simpler to compare algorithms tested using the generic platform with methods presented in papers. While many formats are already recognised by the platform, yet more are in common usage.

During testing the separability metric was found to be less useful than expected. While a good separability does indicate a good algorithm, a poor separability does not show anything except that at least one iris code false match had a Hamming distance that was lower than most of the true matches. Work is needed to either improve or supplement this metric to make it more useful.

It would also be helpful to increase the number of options selectable within the generic platform. For example, normalisation already has a pre-processing module in the form of the segmentation mask, and a post processing module would allow greater experimentation in that area of iris recognition.

The high cost of MATLAB and the consideration of languages such as Python suggest that greater performance and flexibility would result from a conversion of the generic platform's development environment and language away from MATLAB. An alternative language should be investigated with a view to carrying out this conversion.

Appendix A Histogram Thresholding to Find the Pupil Boundary

To find the pupil some researchers [30, 27] find the highest peak in the histogram below intensity 100 and assume that is the pupil. For the CASIA V1 this can be clearly seen in Figure 6.2.

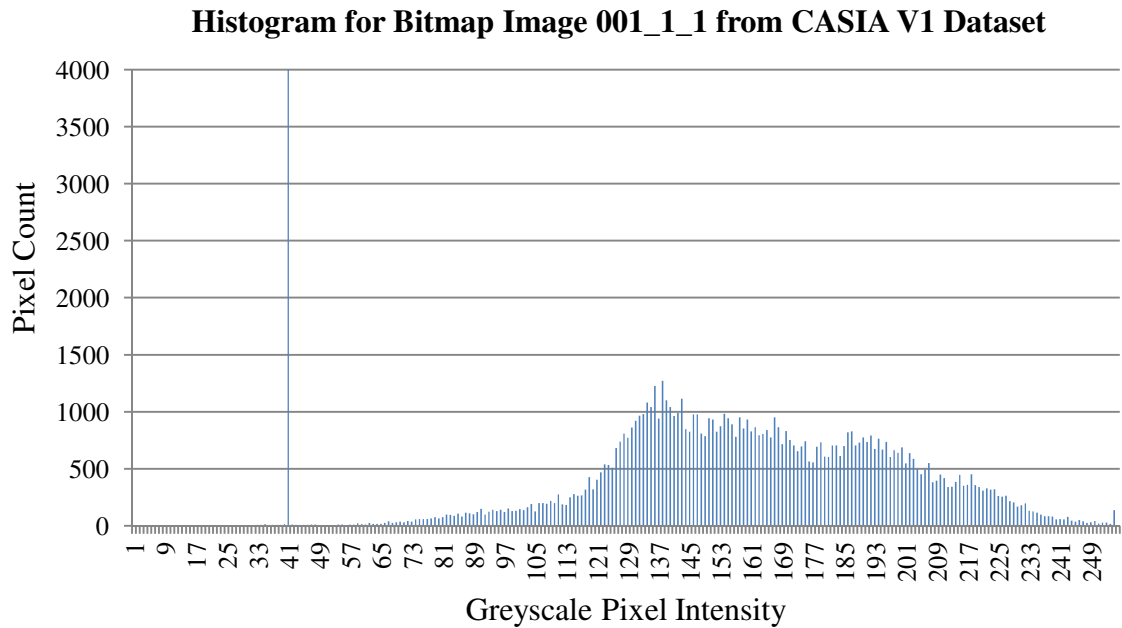


Figure 6.2 – Histogram for bitmap image 001_1_1 from CASIA V1 dataset showing the large peak at intensity 41

The large peak at intensity 41 visible in Figure 6.2 encompasses the pupil modification as well as features such as eyelashes and shadows that normally occur within iris images.

Using the generic platform it was possible to search within the CASIA V3i encoded irises for the original, unedited iris image that 001_1_1 was derived from. Image S1143R01 was the best match according to the generic platform and visual inspection confirmed that this was the unedited image.

This search result enabled the histograms of images 001_1_1 and S1143R01 to be compared directly, this comparison is shown in Figure 6.3

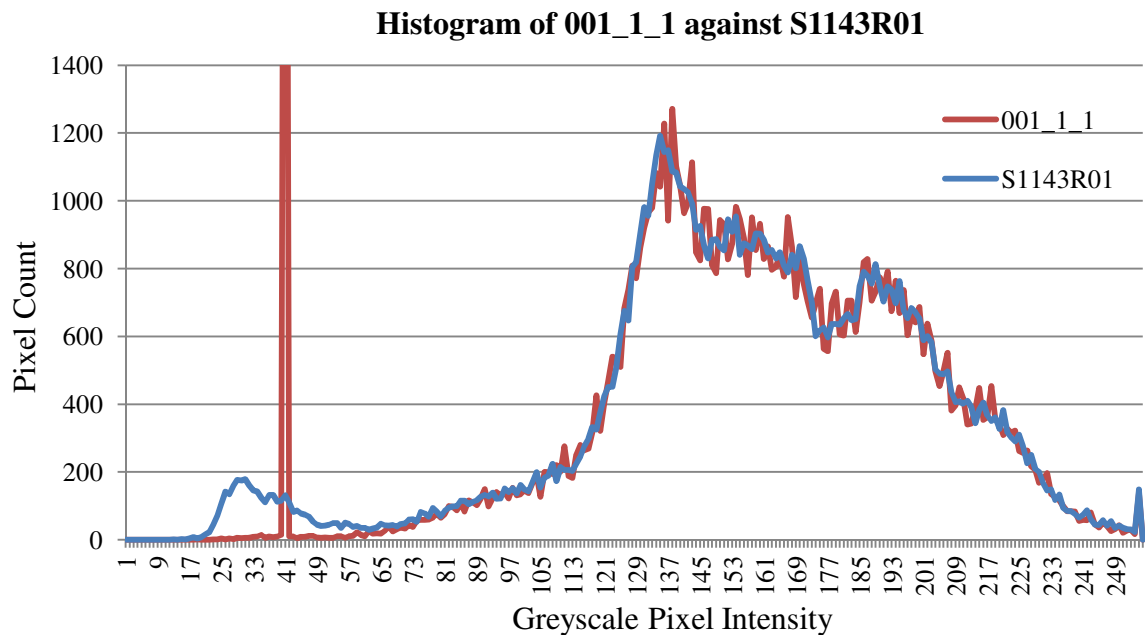


Figure 6.3 – Histogram for image 001_1_1 from CASIA V1 against S1143R01 from CASIA V3i

The 'Pixel Count' (y) axis in Figure 6.3 is truncated at 1400 to prevent the very large peak at intensity 41 from making the rest of the results unreadable. The most significant difference in the histograms can be seen around the base of the pupil spike at pixel intensity 41. The blue histogram of the original image, S1143R01, has a wide undulating lump that starts at intensity 19 and returns to a minima at 63.

The red histogram of the CASIA V1 version of the image, 001_1_1, is very close to a zero pixel count for intensities between zero and 58, with the exception of the large pupil spike at 41. Even after pixel intensity 70, when the two curves approximate, visible differences remain, as the blue curve for S1142R01 looks smoother than the red curve for 001_1_1. All images in the CASIA V1 dataset give a similar results. This shows that the modifications made to the CASIA V1 images by CASIA have affected the entire histogram for the image, not just the low intensity parts.

These algorithms look for the highest pixel count below pixel intensity 100. Figure 6.2 shows that for the CASIA V1 images, the peak histogram count below 100 is very simply determined. Knowledge of the modifications made to the CASIA V1 dataset combined

with the histogram comparison in Figure 6.3 confirms that this obvious peak is, mostly, pupil.

However, results from other image datasets are not as clear-cut. In Figure 6.3, for the blue S1143R01 plot, the peak intensity below 100 occurs at a pixel intensity of 31, but a small extension of the search range from those below 100 to those below 110 would change the maximum to occur at intensity 107. A small change in the lighting or gamma of the capture device could see that curve move down and cause this significant change, rendering any algorithm searching for the pupil ineffective.

It was thought likely that algorithms using this method would have difficulties with datasets where the highest number of pixels below intensity 100 was not a definite spike like the CASIA V1, and where the pupil contained more than one colour. In these unedited images, there is no certainty that the peak intensity in the histogram below 100 is actually the pupil pixels. This expectation was derived from the belief that the threshold method was flawed, as it was based entirely on the CASIA V1 modifications [137].

Section 4.2 discusses thresholding in general and shows that it is not always effective. Due to the nature of this use of thresholding, it is difficult to guarantee that there will be no stray pixels of the same colour (noise) as those being sought (signal), even when using a large spike in the histogram to focus the threshold value.

Appendix B Erroneously Good Results from the Similarity Between Images

It was discovered during that using the whole iris recognition flow results as a guide to the performance of an algorithm was a flawed concept, when checks within the process were not applied. This problem specifically targeted the segmentation stage. Due to the consistency of images of the same eye within some image datasets, it is possible to obtain successful matching using any part of the image, not just the iris, so long as the part selected is consistent.

Figure 6.4 shows some iris images with segmentation boundaries overlaid that were output by Lin *et al.*'s segmentation process.

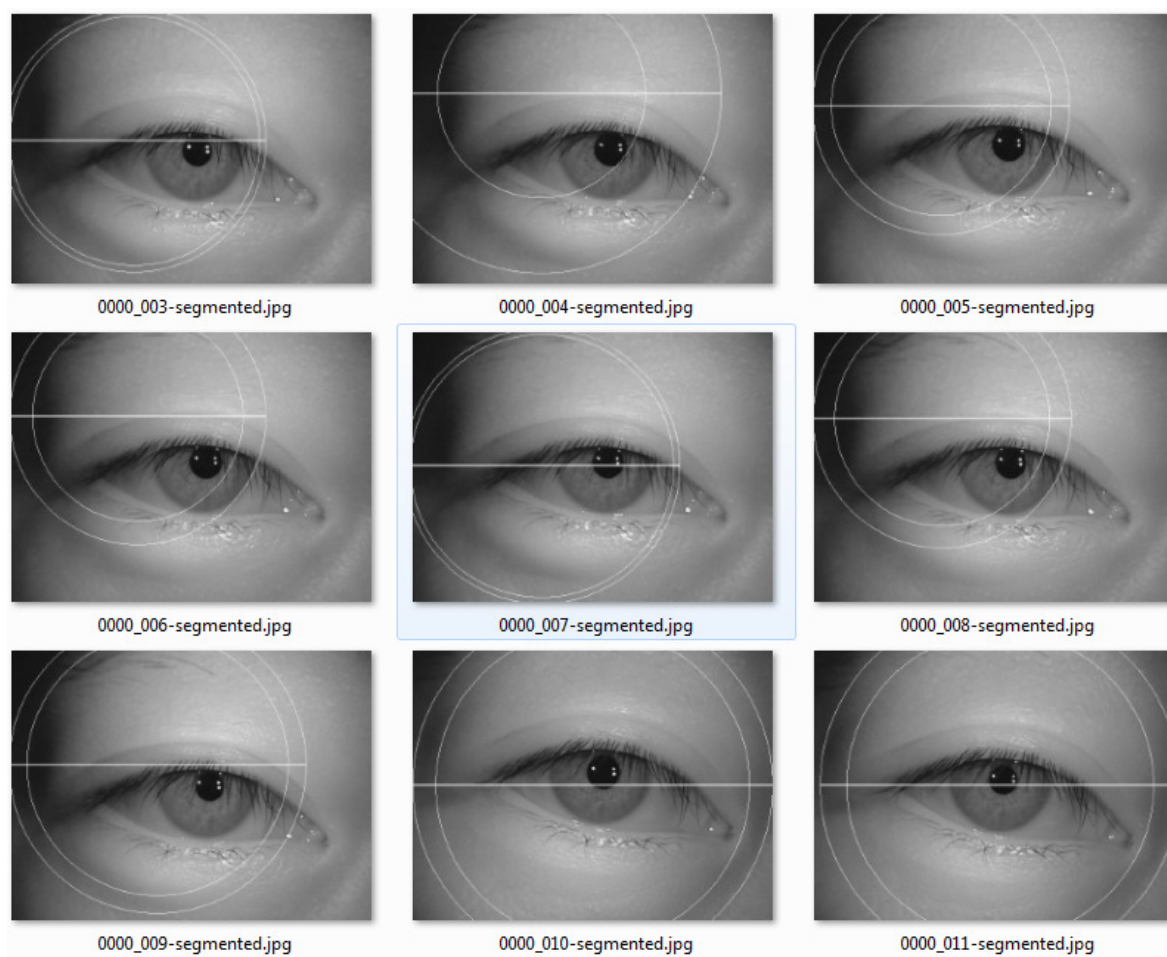


Figure 6.4 – CASIA V2D1 images segmented by Lin *et al.*'s method within the generic platform

Seven of the nine images in Figure 6.4 show iris-pupil boundaries that have their left quadrant in the dark top left corner of the image and their right boundary intersecting with one of the eyelashes. The shadow and the eyelashes look to be similar intensities, therefore, the intensity peak below 100 is likely the shadow and eyelashes, rather than the pupil, though that may also fall within this range. Thus, the cause of the incorrect iris-pupil boundaries is poor intensity threshold discrimination.

Although the segmentation boundaries in the images in Figure 6.4 are nowhere near the iris there are typically two or three images for each class that are very similar, so the segmentation occurs in similar places.

The segmentation boundaries for images 0000_005, 0000_006 and 0000_008 in Figure 6.4 can all be seen to occur at similar locations within the images. Other images whose segmentation boundaries visually appear to coincide completely with each other are 0000_010 and 0000_011. Images 0000_003 and 0000_007 seem to have partially coincidental boundaries in the top quadrant of the circles, though their radii are clearly different.

The coincidental boundaries across images mean that the same parts of the image will be used during the encoding stage. Since the encoding process simply looks for texture and the images are very similar, the skin texture will be encoded and will present a good match. Thus, there will be at least one correct match for many of the images.

This consistency of recognition boundaries between different images of the same eye was due to the images being highly consistent in their lighting and location of the eye and surrounding facial features within the image. This image consistency led to consistent errors in the segmentation algorithm that put boundaries in similar locations on different images of the same eye.

Appendix C The Hough Transform

The Hough transform [41] is a shape detection method that counts the number of pixels in an image that coincide with the shape being sought. The highest counts represent the instances of the shape that coincide with the most pixels and so are the most likely candidates for the location of the shape or object.

The most common uses for the Hough transform are finding circles and lines. Since these are both geometric shapes with straightforward well-known calculations, they both use a similar method to calculate the transform matrix.

The first step in using a Hough transform to find shapes within an image is to create an edge map showing the points of highest rate of intensity change within the image. To maximise the number of useful objects detected, the amount of information in the source picture should be minimised. This is often achieved by smoothing the image with either an averaging or Gaussian smoothing function and by selecting appropriate edge detection thresholds for the application. Thus, the selection of edge detection method, and any threshold values used, has a significant impact on effectiveness of this algorithm.

For the circular Hough transform, the next step is to create the accumulator, this is typically an integer array of the same size as the image, though it is possible to use accumulators that are larger than the image if it is anticipated that the circles that are sought might have their centres located outside the boundaries of the image.

Unfortunately, the Hough transform needs one two-dimensional accumulator for every diameter of circle that is being sought. So, the Hough matrix for circles is actually three dimensional, i.e. an array of two-dimensional accumulators that are each the same size as, or bigger than, the image being evaluated. To search for radii from 50 to 150 pixels within a 640×480 pixel image would require an accumulator that was $101 \times 640 \times 480$, 31 million, integers in size. On modern PC's using 64-bit integers, that would require 248 million bytes of memory.

Once the accumulator is created, the algorithm will search for all the edge points within the edge image. For each point in the edge image, the Hough transform will draw a circle on every one of the two-dimensional arrays within the accumulator. Each one of these circles

will have their radius defined by the size of circle being sought. So, for the previous example of searching for circles between radii of 50 to 150, the Hough transform will draw 101 circles in the accumulator.

On the first two-dimensional accumulator element, a circle of radius 50 pixels will be drawn at the x, y coordinates of the first edge point. The second two-dimensional accumulator element will have a circle of radius 51 drawn upon it; the third will have a circle of radius 52 drawn upon it; and so on up to the last two-dimensional array element which will have a circle of radius 150 drawn upon it.

This process is repeated for every edge point within the edge image. Therefore, for a 640×480 point edge image with 100,000 edge points, 10.1 million circles would be drawn within the accumulator. However, as the name accumulator may suggest, this is not an image that is being drawn, so the drawing process is a little more complicated than simply setting each pixel on the circumference of the circle.

The Hough transform works because every pixel found adds one to the centre location of every possible circle that pixel could lie on. Therefore, when the Hough transform is drawing a circle it is actually adding one to the centre locations for each of the circles whose circumference intersects with the currently selected edge point.

The more times a location within the accumulator is included as a potential circle centre, the higher the count at that location becomes.

When all the circles for all the edge points have been drawn in the accumulator, it will resemble Figure 6.5.

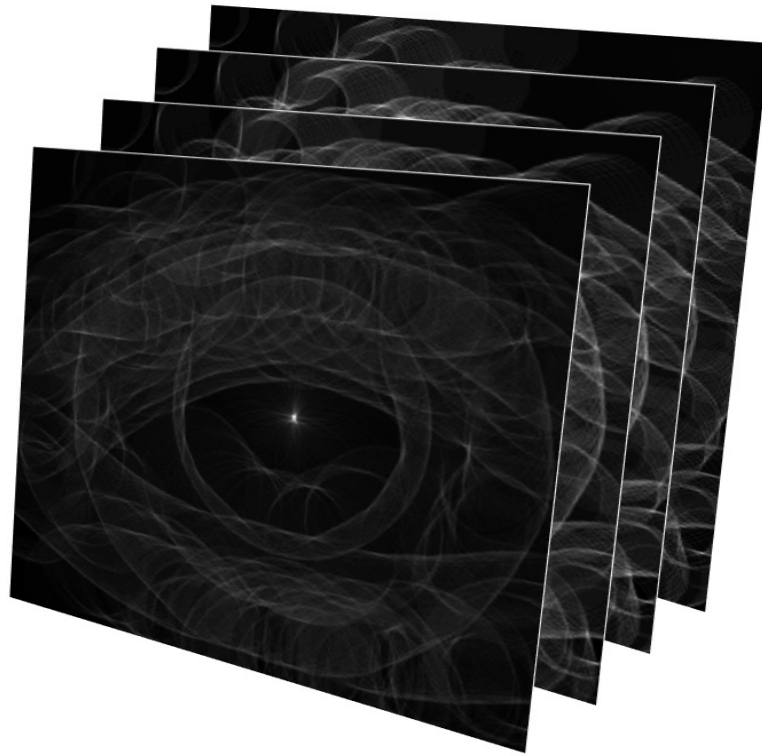


Figure 6.5 – Four accumulators from the three dimensional Hough matrix created by segmenting the CASIA V1 image 001_1_1.

Figure 6.5 shows an accumulator that has been converted to converted to grey scale images and is finding four radii of circle. These particular images are taken from the accumulator for Segment One segmenting the CASIA V1 image 001_1_1. Each one of the four images is searching for a specific circle radius. The whiter the pixel in the image, the higher the accumulator value is at that point, and the stronger the circle candidate is.

The location within each of the four accumulators that has the highest value is the centre point of the strongest circle of that size. The four maxima are then compared and the highest one of the four determines the radius of the circle.

Figure 6.6 shows the top two accumulators side-by-side. Figure 6.6a shows an accumulator with lots of circle candidates and several brighter white maxima. Figure 6.6b shows a single strong white maxima in the centre of the image, indicating a potential circle centre candidate.

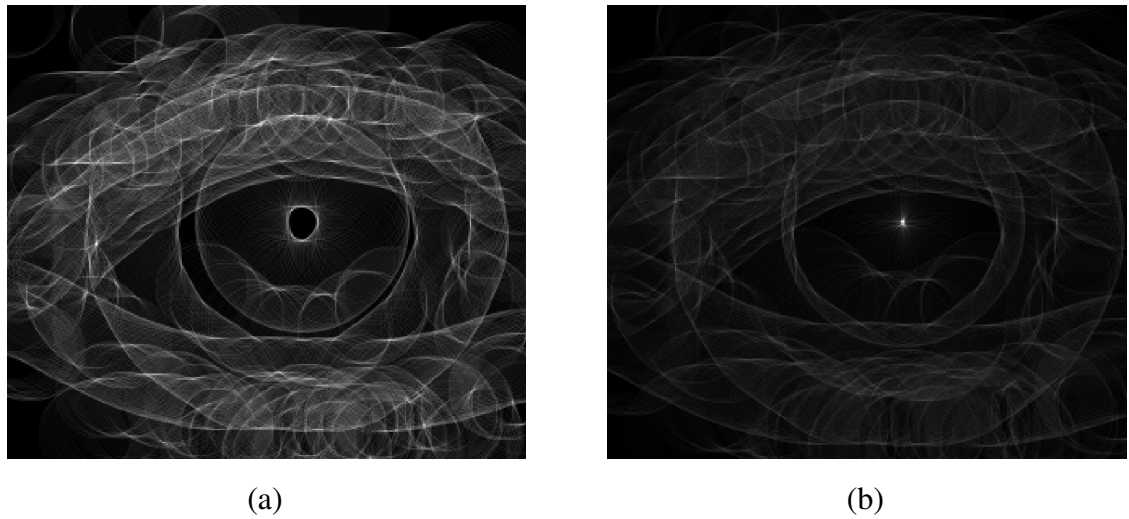


Figure 6.6 – Two accumulators (a) showing widely disparate maxima, (b) showing a focussed maxima at the centre

The example accumulators in Figure 6.5 and Figure 6.6 drew 452,081 circles and took 12.5 seconds. For this reason a fast circle algorithm is needed for implementing a Hough transform. The algorithm used for the implementation of the Hough matrix for Segment One was a direct translation of [158] from C into MATLAB.

The Hough transform process as pseudo code gives:

```
// Get the image information
Get image size into ImageY and ImageX variables

// Create the circle radius parameters
Let dRadius = the number of radii to search for
Let minRadius = the smallest radius to search for

// Create the empty Hough Matrix
Let HoughMatrix = Empty Array of (ImageY × ImageX × dRadius)

// Get the edge image point information
Let PointList = the list of edge points in the edge image
Let iNumPoints = the number of edge points listed
```

```
// Draw the circles
for iPoint = 1 to iNumPoints // One point at a time
  for iRadius = 1 to dRadius // One circle at each radius
    Radius = minRadius + iRadius - 1
    DrawCircle(HoughMatrix(iRadius), PointList(iPoint), Radius)
  endfor
endfor

// Find the brightest point
for iRadius = 1 to dRadius
  Let iMax(iRadius) = Maximum(HoughMatrix(iRadius))
endfor

// Find the strongest circle
Let Circle = Maximum(iMax)
```

This pseudo code is highly simplified but demonstrates the basic process. Information about the image and circles is gathered. The accumulator is then created and populated with the circles. The strongest circle is then found by looking for the highest value within the accumulator.

Appendix D The Circle Bisector Method

The perpendicular bisector method, Figure 6.7, is the traditional algebraic method for calculating the radius and centre location of a circle, from three points on the circumference.

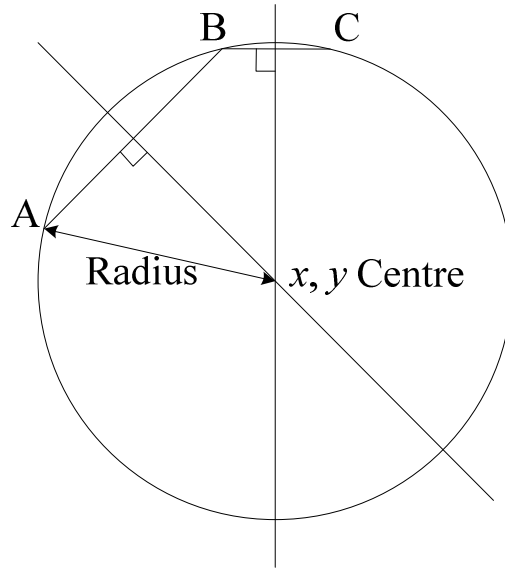


Figure 6.7 – Perpendicular Bisectors

The three circumference points are then used to calculate the circle centre using the circle bisector equations 6.1, 6.2 and 6.3 to calculate the centre and radius of a potential circle.

$$x_o = \frac{\left(\frac{A_x - B_x}{B_y - A_y}\right)\left(\frac{A_x + B_x}{2}\right) - \left(\frac{A_y + B_y}{2}\right) - \left(\frac{A_x - C_x}{C_y - A_y}\right)\left(\frac{A_x + C_x}{2}\right) + \left(\frac{A_y + C_y}{2}\right)}{\left(\frac{A_x - B_x}{B_y - A_y}\right) - \left(\frac{A_x - C_x}{C_y - A_y}\right)} \quad 6.1$$

$$y_o = \left(\frac{A_x - B_x}{B_y - A_y}\right)\left(x_o - \left(\frac{A_x + B_x}{2}\right)\right) + \left(\frac{A_y + B_y}{2}\right) \quad 6.2$$

$$r = \sqrt{(A_x - x_o)^2 + (A_y - y_o)^2} \quad 6.3$$

where (A_x, A_y) , (B_x, B_y) and (C_x, C_y) are the three edge image points, (x_o, y_o) are the centre coordinates for the circle and r is the calculated radius of the circle.

The perpendicular bisector method is remarkable in that it uses very simple geometry to find the correct answer. This means that the resulting algorithm is simple and fast.

One of the problems encountered when using the circle bisector method in a computer is that the boundary points have been digitised, and this quantisation of the data leads to errors in calculations. Therefore, using the points of a circle perimeter to calculate its centre will return several different locations and radii. Even worse, if adjacent points are used then the result will be many very small circles.

Appendix E Published Paper, ICB 2012

The following paper was published at ICB in March 2012.

A Generic Computer Platform for Efficient Iris Recognition

Chris Ponder
 Institute of System Level Integration (ISLI)
 Heriot-Watt University Research Park
 Research Avenue North
 Edinburgh EH14 4AP
chris.ponder@sl-i-institute.ac.uk

Khaled Benkrid
 The University of Edinburgh,
 Institute of Integrated Systems,
 Edinburgh EH9 3JL,
k.benkrid@ed.ac.uk

Abdsamad Benkrid
 University of Portsmouth,
 School of Engineering, Portsmouth PO1 3DJ,
abdsamad.benkrid@port.ac.uk

Abstract

This paper presents a new open source MATLAB based generic platform for the development, testing and evaluation of iris recognition algorithms. This platform uses industry standard image datasets and test methodologies to evaluate the performance of iris recognition algorithms and output consistent, comparable and quantifiable results. The results generated by the platform include industry standard FAR vs FRR, ROC and EER as well as true match, false match and separability. This improves the ability to determine the efficacy of a given algorithm or variation.

This presentation shows existing open source algorithms being adapted to work within the platform and then evaluated using the platform with openly available iris image datasets. The simplicity and speed of adaptation and the comparability of the results produced is clearly demonstrated. Furthermore, the benefits of using the platform for algorithm development and improvement are highlighted by demonstrating the platform's ability to interchange algorithms, or parts of algorithms with the click of a mouse.

1. Introduction

This paper discusses an open source generic platform to develop, test, evaluate and benchmark new and existing recognition algorithms. The platform is flexible yet consistent, open and easy to extend yet provides a base for evaluating algorithms producing repeatable, quantifiable results. The platform follows the process, as defined by Daugman, of recognising and matching an iris pattern

First, some brief background information and the state of the art is given. Following this the platform and then some results from implemented algorithms are presented. Finally, the conclusions and future work are discussed.

2. Background

Iris recognition is considered to be one of the most accurate biometric methods available for the unique identification of individuals. With many international border controls using iris recognition to verify individual's identities its use is increasing. Most commercial systems are based around the methods originally developed by Dr John Daugman of Cambridge University [76, 159]. Yet there is very little in the way of consistent independent testing of these systems, very little in the way of independent comparative data between the various techniques employed to do the analysis and no platform for the development, testing and analysis of recognition algorithms.

Libor Masek produced one of the first widely known open source implementations in 2003 [160]. Masek released a set of routines in MATLAB implementing an iris recognition system using Daugman's process. Masek's code is optimised for the freely available CASIA 1 database with adaptations available for the commercial Lions Institute database.

While there are many working in the area of new algorithm development, there seems to be little attempt to provide a platform for testing and analysis. There are, however, a number of open source iris recognition projects:

- CreateIrisTemplate by Masek [160, 29]
- JIRRM <http://sourceforge.net/projects/jirm/>
- The iris recognition DSP library <http://sourceforge.net/projects/irisrecognition/>
- IRIS-ICT by Antonino [161]
- Iris, by Professor Xin Li, of West Virginia University (WVU) [162]
- OSIRIS [163]
- GIRIST <http://www.grusoft.com/girist.htm>

None of these is aiming to be a generic platform and investigations so far have found no evidence that such a platform is being developed elsewhere.

3. The Generic Platform

The design goals of the generic platform are to make development and comparison of recognition algorithms faster, easier and more consistent. The core target process for the modular platform was originally defined by Daugman [1] and is outlined in

Figure 8.

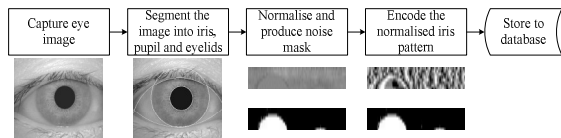


Figure 8 – A generic iris-code capture process

While these steps are central to the process of iris recognition, the output at each stage can be very different. In the segmentation stage where Daugman's latest work uses splines for the eyelid delineation, Masek uses lines. Where Daugman normalises to a rectangle, Birgale and Kokare [8] suggest a method using the original circular iris. Daugman's iris code is 256 bytes, Masek uses 1200 bytes and Birgale & Kokare use 24 bytes. Potential results differences from each stage mean that the platform is designed to cope with all these results and pass them to the next stage, assuming the next stage supports that data type.

The following terms describe the various elements within the generic platform:

- **Image Dataset** – A collection of images stored in a directory on the computer's hard drive, for the purposes of recognition evaluation.
- **Full Flow** – This self-contained iris recognition process takes an image as its input and returns the segmentation, normalisation and encoding results. A full flow will normally contain a second executable that can perform matching on the results returned by the full flow.
- **Custom Flow** – This is a partial or complete iris recognition process broken into four distinct executables or stages designed for the generic platform. Custom flows allow different segmentation, normalisation, encoding and/or matching methods to be selected for evaluation.
- **Stage** – This refers to the iris recognition steps, segmentation, normalisation, encoding and matching.
- **Module** – A module is a group of flows and custom flows, typically but not always by the same author, that add recognition methods to the generic platform.
- **Sub-Module** – These support-functions within a module do not perform an entire flow or recognition stage but are selectable within the GUI. Examples would be edge, circle and line detection as well as segmentation mask

generation.

The platform provides the ability to mix and match recognition stages and sub-module functions using different methods with the click of a mouse. Once the stages and sub-modules are selected, the custom flow selected can be run against a single image or an entire dataset of images. The single image or manual flow will display the output of the various stages and sub-modules once the flow has completed. The dataset or automated flow produces data on the performance of the algorithm across the entire dataset, segmenting, normalising and encoding each image. Each iris code is then compared to all the others in the dataset producing a list of match results with true or false matches determined by file name. This data is then analysed to provide standard results.

The platform has two user selectable modes of operation, automatic and manual. In automatic mode the selected flow is used on an entire image dataset and the results analysed. In manual the platform processes just one image and displays diagnostic information and results for that image. Figure 3.12 shows the generic platform GUI in manual mode with each area of interest labelled with a large blue number.

1. The user selectable list of image datasets
2. The user selectable list of results datasets
3. The user selectable output options for both automatic and manual modes
4. The user selectable process flow method
5. The user selectable list of images in the currently selected image dataset
6. The currently selected image with its histogram
7. File details for the currently selected image
8. The custom selection and results area
9. The process style selection and initiation
10. Function selection for area 8

Area (8) allows algorithms from different researchers and developers to be selected, providing a simple click selection from lists to change the algorithms used. This is one of the core features of the generic platform, making it incredibly flexible and straightforward to use. This flexibility goes further than just switching core algorithms for segmentation, normalisation, encoding and matching. If the algorithm is coded to support it, the user can switch out sub functions, for example for segmentation, different edge, circle or line detection can be used to see the effect of these on the recognition rates. These settings can be configured for each of the four boundaries (pupil, iris, upper and lower eyelid) being detected. The 'Try it' buttons under each list box allow the selected algorithm to be tested without the need to invoke the entire flow.

Images displayed in area (8) of the GUI can be opened in a separate figure window by clicking on the small button labelled 'F' in their top left corner. This allows the images to be resized and magnified

as required.

Extending the platform is a simple process, made even more so by the nature of MATLAB coding. Datasets are added by placing them in their own subdirectory off the main image directory. Modules, flows, stages and sub-modules are added by placing them in their own 'mod_' subdirectory off the main program directory with an initialisation '.m' file.

All of the platform list boxes are populated by the module code or by searching datasets. During the first run of the platform the user is asked to specify the root folders for the image datasets and the results database, this is then saved to a file in the main program directory as <HOSTNAME>.opt. The code then scans the program directory for folders beginning with 'mod_', a MATLAB '.m' file with the same name as the directory is then sought within each 'mod_' directory. The '.m' file contains all the configuration information for the given module.

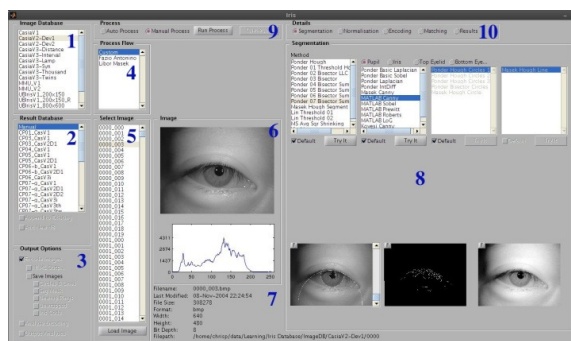


Figure 9 – Generic Platform GUI with 'Manual Process' selected and GUI regions numbered in blue

4. Testing Overview

There are several criteria used for measuring the performance of iris recognition processes [164] that are implemented on our generic platform:

- True Match – Two eye images that are confirmed as being the best match and from the same physical eye
- False Match – Two eye images that are confirmed as the best match but are *not* from the same physical eye
- True Match Rate – The percentage of best matches for a given flow and dataset that are true matches
- False Match Rate – The percentage of best matches for a given flow and dataset that are false matches
- Separability. Sometimes called discernability, this is The percentage of true matches with a Hamming distance lower than the lowest false match for a given flow and dataset
- FAR, False Accept Rate – This is a measure of the percentage of irises that are accepted for

being a match which are from different physical eyes

- FRR, False Reject Rate – This is a measure of the percentage of irises that are detected as being from different eyes when they are in fact images from the same physical eye
- EER, Equal Error Rate – This is the percentage of FAR and FRR when the FAR is equal to the FRR

The FAR and FRR are not absolute figures but show the change in the number of false accepts and rejects over the range of Hamming distances that contain both true and false matches. True and False matches are only possible where the physical eye that two images come from can be determined computationally. In the case of image datasets there are typically consistent naming conventions that allow this to be determined. These results are all returned automatically assuming that the user selects the analysis output option.

The machine used for testing was a Dell Latitude Core2Duo T9900 at 3.06GHz with nVidia Quadro NVS 160M and 4GB RAM running 64-bit LinuxMint 10 (Gnome Ed.) and MATLAB r2009a.

While this research does not look at different capture devices, it does use pictures from several different sources, produced using a variety of devices, stored in both uncompressed bitmap and compressed JPEG2000 image formats. The platform supports as many eye image databases as the user has available, the following datasets were used during the course of this research:

- **CASIA**

The Chinese Academy of Sciences' Institute of Automation (CASIA) datasets are used throughout the world and have been referenced in many publications [159, 30, 160]. They contain thousands of images for general research purposes. Each image is an 8-bit grey-scale image ranging in size from 320x280 pixels up to 640x480 pixels.

<http://biometrics.idealtest.org/>

- **MMU**

There are two MMU iris datasets; each collected using a different camera. The MMU datasets offer a good selection of ethnicity and quality of image to stress the algorithm under test but a low image count when compared to CASIA.

<http://pesona.mmu.edu.my/~ccteo/>

- **UBIRIS**

The UBIRIS 1 dataset is much like the MMU and main CASIA datasets in providing consistent close-up images of eyes provided in two different resolutions as well as colour and greyscale. The UBIRIS 2 dataset provides a

much wider variety of image qualities and distances than the other iris image datasets.

<http://iris.di.ubi.pt/ubiris1.html>

<http://iris.di.ubi.pt/ubiris2.html>

- **WVU Sample**

The WVU sample dataset is a small subset of the full WVU image dataset consisting of 80 images. This dataset is downloadable without registration

<http://www.csee.wvu.edu/%7Exinl/code/iris.zip>.

Since this paper is presenting a method of developing and benchmarking iris recognition algorithms, it is the consistency and usefulness of the results that matters. Thus, a single image dataset was used for the testing presented. The CASIA V1 dataset was used, despite its shortcomings [165], as it was the dataset used to develop the algorithms being implemented.

5. Use Cases

In order to assess the performance and suitability of the generic platform it was put through several distinct scenarios.

5.1. Adding an Image Dataset

A new image dataset (WVU sample set) was copied into folder “WVU” within the user selected image database folder and the GUI restarted. “WVU” then appeared in the ‘Image Database’ listbox, Figure 3.12 section (1).

5.2. Adding an Existing Flow

Two existing flows with available MATLAB code, Masek [29] and Antonino [161], were identified. Antonino’s flow uses Masek’s segmentation and normalisation but different encoding and matching. To add a flow requires three lines of code to be added to the ‘mod_*.m’, e.g. mod_LiborMasek.m:

```
function Info = mod_LiborMasek
    iFlow = 1;
    Info.Flow(iFlow).Name = 'Libor Masek';
    Info.Flow(iFlow).Cmd = 'createiristemplate';
```

‘iFlow’ is simply an index, this can make adding more entries very straightforward. ‘Info.Flow()’ is the return structure for flows, with ‘Name’ as the displayed name and ‘Cmd’ the name of the m-file to execute. The top level function must then conform to the I/O calling convention:

```
Results = CreateIrisTemplate(EyeImage,
                             bDebugImages)
```

Where EyeImage is the greyscale eye image array to be processed, bDebugImages is a Boolean indicating if the calling routine wants the debug images created and Results is a structure which

contains all the results. The name of the routine can be anything the researcher/developer desires as it is defined in the module configuration file.

The analysis module requires a Matching module to allow it to compare iris-codes. The matching module takes a Record and compares it to all the records in the encoded database (MatchDB).

```
Results = Match(MatchDB, Record, bAvoidSelf,
                Options)
```

The routine can be told to avoid comparing ‘Record’ to files with the same filename (bAvoidSelf) and any ‘Options’ required can be passed to it. ‘Results’ is a structure containing the results of the matching process. The Match file must also be added to the module initialisation file for it to be available within the platform.

```
iMatch = 1;
Info.Match(iMatch).Name = 'Masek Hamming';
Info.Match(iMatch).Cmd = 'Im_match';
Info.Match(iMatch).Style = 'Masek HD';
```

Conversion to the platform, including debug time, was completed in less than 90 minutes for each flow. Both flows were then able to take full advantage of the analysis routines and image databases available via the platform.

5.3. Customising an Existing Flow

Customising a flow to work within the framework of the platform involves breaking the flow into four stages: segmentation, normalisation, encoding and matching. These four stages have consistent interfaces:

```
Results = Segment(ImageIn, bDebugImages,
                  Options)
Results = Normalise(ImageIn, bDebugImages,
                   Options)
Results = Encode(ImageIn, Options)
Results = Match(MatchDB, Record, bAvoidSelf,
                Options)
```

Where ‘ImageIn’ is a structure containing all the basic information about the current image at each stage, consistent with the ‘Results’ structure. Splitting the two flows into stages took less than two hours in total.

5.4. Adding Sub-Module Configurability

While the ability to switch out Masek’s segmentation algorithm for a different one is useful, being able to, for example, change the edge detection or Hough matrix used by Masek’s circle detection code allows for some very interesting experimentation. The platform allows edge, circle and line detection routines to be customised, where supported in code, for the pupil, iris and both eyelids individually. The segmentation mask used can also

be selected via the GUI under the normalisation 'tab'. This allows a considerable number of 'what if' scenarios to be played out with a few clicks of the mouse.

Sub-modules are *not* available to full flows, only to segmentation and normalisation stages. Adding this functionality to a module is a fast process. For this paper, the circle detection sub-module is shown. The findcircle routine call is replaced with a call to the ModuleRun routine.

```
IrisCircleResults =
ModuleRun('FindCircle', EyeImage, Options.Iris);
```

Then the default function is added to the module file:

```
iCircle = 1;
iDefCircle = iCircle;
Info.Circle(iCircle).Name = 'Masek Hough
Circle';
Info.Circle(iCircle).Cmd = 'lm_findcircle';
```

The default circle routine is set in the segmentation section of the module file, this is a text string so can also be a routine from a different module:

```
Info.Segment(iSegment).Iris.CircleMethod =
Info.Circle(iDefCircle).Name;
Info.Segment(iSegment).Pupil.CircleMethod =
Info.Circle(iDefCircle).Name;
```

This process took 20 minutes to complete.

5.5. Adding a New Method

The new method selected for this was Lin's segmentation proposal. A new module directory 'mod_Lin' was added to the platform root and the 'mod_Lin.m' file was created. 'mod_LiborMasek.m' was used as the prototype for 'mod_Lin.m'. Masek's 'lm_segment.m' was copied to the 'mod_Lin' directory and renamed 'lin_segment01.m'. The Masek algorithm was removed leaving just the input and output structures. Once the platform had been restarted, no further restarts were required. Lin's algorithm was now available for selection in the list of segmentation methods. Time required to create the new module and be ready to start developing the new method was ten minutes.

Lin's proposed segmentation method was then coded. This is a fast iterative process of coding, saving the changes and then clicking the 'try it' button to evaluate and debug. This entire process, including debug time, took approximately 2 hours 20 minutes.

5.6. Use Case Results

The use cases show just how quickly and easily the platform can be extended with additional algorithms and image datasets. When compared to

manually writing the code for every aspect of the process the generic platform is significantly simpler and faster.

Having added the Masek and Antonino flows to the platform they were run using the CASIA V1 dataset. The results of these runs are shown in Table 4 and Table 5 and demonstrate how clear the generic platform results are, even from two different methods. It is trivial to compare the two algorithms and see that Antonino's code is faster at matching but less accurate than Maseks.

Table 4 – Recognition performance of the listed segmentation methods run as full flows with the 756 images of the CASIA V1 dataset

Method	True Match	False Match	Separability	EER
Masek	98.0%	2.0%	91.5%	4.1%
Antonino	54.8%	45.2%	13.0%	30.1%

Table 5 – Total recognition times of the listed segmentation methods run as full flows with the 756 images of the CASIA V1 dataset

Method	Encode Time	Analysis Time
Masek	2h 07m	1h 27m
Antonino	2h 09m	0h 52m

Converting the Masek and Antonino flows from monolithic flows to run as modules within the custom flow framework demonstrated that the platform adds no appreciable time or accuracy overheads to the process, though there is some time variation due to operating system load levels. The recognition results in the first two lines of Table 6 and Table 7 are identical to Table 4 and Table 5 respectively, confirming that there is no loss of accuracy. Line three of Table 6 and Table 7 shows the benefits of modularisation, with the Antonino encoding and matching modules being used with scheme two's [12] segmentation code, improving speed and accuracy.

Table 6 – Recognition performance of the listed segmentation methods on the 756 images of the CASIA V1 dataset

Method	True Match	False Match	Separability	EER
Masek	98.0%	2.0%	91.5%	4.1%
Antonino	54.8%	45.2%	13.0%	30.1%
Antonino-Scheme 2	81.1%	18.9%	47.8%	22.1%

Table 7 – Total recognition times of the listed segmentation methods run as custom flows

Method	Encode Time	Analysis Time
Masek	2h 08m	1h 19m
Antonino	2h 07m	0h 53m
Antonino-Scheme 2	0h 08m	0h 50m

Table 8 and Table 9 accentuate the benefits and speed of using sub-modules. The Masek and Antonino routines were run with the segmentation code from Masek using a faster Hough routine coded by this researcher. This was selected from the GUI using just three mouse clicks and shows a dramatic speed increase. Table 8 shows a slight reduction in true/false match performance but an improved separability for both algorithms. The 80% reduction in ‘Time 1’ (Table 9) is quite considerable; demonstrating that Hough transform performance is a significant bottleneck for the segmentation code by Masek.

Table 8 – Recognition performance of the listed segmentation methods using an alternate Hough circle sub-module with the 756 images of the CASIA V1 dataset

Method	True Match	False Match	Separability	EER
Masek	97.8%	2.2%	93.2%	4.4%
Antonino	53.8%	46.2%	40.5%	29.7%

Table 9 – Total recognition times of the listed segmentation methods using an alternate Hough circle sub-module with the 756 images of the CASIA V1 dataset

Method	Encode Time	Analysis Time
Masek	0h 21m	1h 19m
Antonino	0h 23m	0h 52m

The platform also produces industry standard ROC and Far vs FRR graphs in JPG and EMF (on Windows™) picture formats.

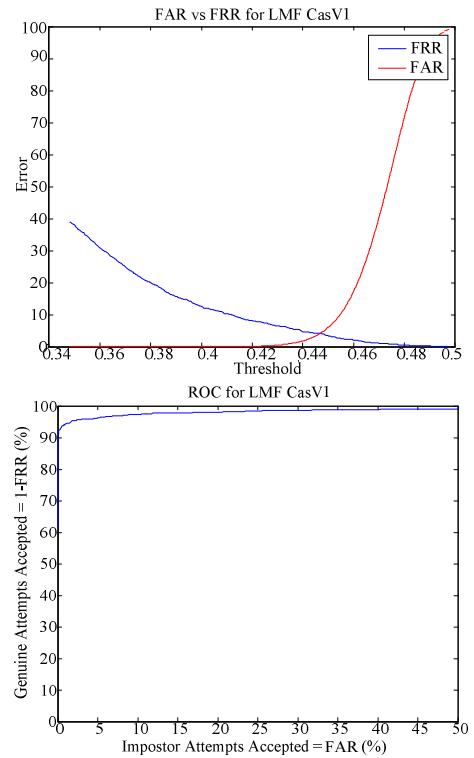


Figure 10 – FAR vs FRR and ROC for Masek full flow

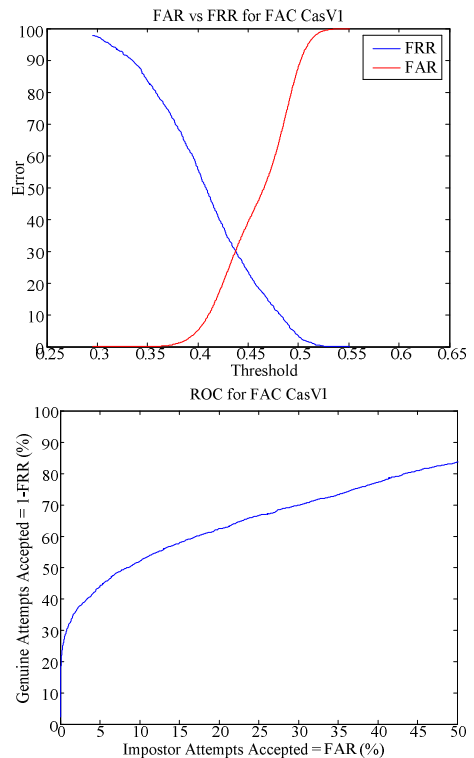


Figure 11 – FAR vs FRR and ROC for Antonino Custom flow

6. Conclusions

This paper presented a unique open-source generic platform for the development and consistent evaluation and comparison of iris recognition algorithms. The research clearly shows the simplification and increased development speed of algorithm development for the encoding and analysis of iris recognition algorithms provided by our generic platform. Without the need to implement initialisation, debug, output, test and analysis code the development can focus on the recognition stage or flow being implemented.

The platform has been used by us to test existing open source code from Masek [160], Antonino [161] and Li [162], implement methods proposed in papers by Lin [10] and Shamsi [167], and by Ponder to develop a new segmentation method [12].

Industry standard results EER, FAR vs FRR and ROC graphs [164], are produced by the platform to allow direct comparison with previously released methods and commercial systems.

There is still work to be done including adding the ability to automatically ascertain the accuracy of segmentation with some degree of confidence.

More open source flows will be added, as well as more normalisation, encoding and matching methods. HTML output is also planned so that text can be combined with the results graphs. It is planned that the platform will be released under an open source license at the conclusion of this research.

7. Bibliography

- [1] J. Daugman, "How iris recognition works," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 21-30, Jan. 2004.
- [2] A Bouridane, "Recent Advances in Iris Recognition: A Multiscale Approach," in *Imaging for Forensics and Security*.: Springer Science+Business Media LLC, 2009, ch. 4, pp. 49-77.
- [3] Libor Masek, "Recognition of Human Iris Patterns," The University of Western Australia, BEng Thesis 2003.
- [4] Libor Masek. (2010, July) School of Computer Science and Software Engineering : The University of Western Australia. [Online]. <http://www.csse.uwa.edu.au/~pk/studentproject/s/libor/>
- [5] Fazio Giacomo Antonino. (2011, Apr.) Iris Recognition. [Online]. <http://sourceforge.net/projects/iris-ict/>
- [6] Xin Li. (2011, Aug.) Xin Li's Homepage. [Online]. http://www.csee.wvu.edu/~xinli/demo/nonideal_iris.html
- [7] Inmaculada Tomeo-Reyes, Judith Liu-Jimenez, Ivan Rubio-Polo, Jorge Redondo-Justo, and Raul Sanchez-Reillo, "Input images in iris recognition systems: A case study," in *Systems Conference (SysCon), 2011 IEEE International*, 2011, pp. 501-505.
- [8] Lenina Birgale and M Kokare, "Iris Recognition Without Iris Normalization," *Journal of Computer Science*, vol. 6, no. 9, pp. 1042-1047, 2010.
- [9] J Daugman, "Biometric decision landscapes," Computer Laboratory, University of Cambridge, Cambridge, ISSN: 1476-2986, 2000.
- [10] W T Lin, C C Liu, and S Y Chen, "Fast and robust iris recognition," *The Imaging Science Journal*, vol. 57, no. 3, pp. 128-139, June 2009.
- [11] P.J. Phillips, K.W. Bowyer, and P.J. Flynn, "Comments on the CASIA version 1.0 Iris Data Set," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 10, pp. 1869 - 1870, Oct. 2007.
- [12] Chris Ponder. (2011, Sep.) A Novel Bisector Approach to Iris Recognition. PDF.
- [13] M. Shamsi, P.B. Saad, S.B. Ibrahim, and A.R. Kenari, "Fast Algorithm for Iris Localization Using Daugman Circular Integro Differential Operator," in *International Conference of Soft Computing and Pattern Recognition*, 2009, pp. 393-398.

Appendix F Filenames of Images Removed from CASIA V1

The following list defines which images were removed from the CASIA V1 in an attempt to replicate Masek's results in Section 4.1. The first list is the images removed from the CASIA V1 dataset to create a reduced dataset with 633 images.

002_1_1	002_1_2	002_1_3	004_2_2	005_2_1	006_1_1
006_1_2	006_1_3	006_2_1	006_2_2	006_2_3	006_2_4
007_2_3	011_1_3	011_2_3	019_1_1	019_1_2	019_1_3
019_2_2	021_2_2	026_2_2	027_2_3	030_1_3	037_1_1
037_1_2	037_1_3	037_2_1	037_2_2	037_2_3	037_2_4
039_2_1	041_1_1	041_1_2	041_1_3	041_2_1	041_2_2
041_2_3	041_2_4	046_1_1	046_2_4	048_1_2	049_1_2
049_2_1	049_2_2	049_2_3	049_2_4	050_1_1	051_2_4
052_1_1	052_1_2	052_1_3	052_2_1	052_2_2	052_2_3
052_2_4	054_2_2	054_2_3	055_2_2	058_1_1	058_1_2
058_1_3	061_1_1	061_1_2	061_1_3	061_2_2	063_2_4
065_2_3	066_1_1	067_1_1	070_1_1	070_1_2	070_1_3
070_2_4	077_1_3	077_2_1	078_1_2	078_2_1	078_2_3
079_1_2	081_1_2	086_1_1	086_1_2	086_1_3	086_2_1
086_2_2	086_2_3	086_2_4	091_1_2	091_2_1	094_1_1
094_1_2	094_1_3	094_2_2	095_1_1	095_1_2	095_1_3
095_2_1	095_2_4	096_2_1	099_1_3	101_1_1	101_1_2
101_1_3	101_2_1	101_2_2	101_2_3	101_2_4	104_2_1
104_2_2	104_2_3	104_2_4	106_1_1	106_1_2	106_1_3
106_2_1	106_2_2	106_2_3	106_2_4	107_2_1	107_2_2
107_2_3	107_2_4	108_2_3			

The following list defines the remaining nine images that were removed from the remaining 633 images of CASIA V1 in order to match Masek's count of 624 images.

001_1_1 002_2_1 003_2_4 004_1_2 027_2_2 039_1_1
051_1_1 051_1_2 051_1_3

Appendix G Example Analysis Output

Output started: <Date & Time>

Summary Statistics

=====

Database:

Image Database: <Name>

Result Database: <Name>

Recognition:

True Match: <>% (TM / Max)

False Match: <>% (FM / Max)

True Match below False Match minimum: <>%

Equal Error Rate (EER) <>%

EER Hamming Threshold <>

Segmentation:

Pupil: <>%

Iris: <>%

Top Eyelid: <>%

Bottom Eyelid: <>%

Average: <>%

Speed:

Average Segment time per record: <Time> seconds

Average Normalise time per record: <Time> seconds

Average Encode time per record: <Time> seconds

Total Encoding time per record: <Time> seconds

Total encoding time: <Time>

Total analysis time: <Time>

System Info:

Host Name: <Name>
CPU: <Name>
CPU Clock: <MHz>
Number of Processors: <Count>
System RAM: <MB/GB/TB>
OS Version: <Name>
OS Architecture: <Name>
MATLAB Version: <Version>

Source Database: <Name>

=====

Flow: <>
Segmentation:<Name>
 Pupil Edge Detection: <Name>
 Iris Edge Detection: <Name>
 Top Eyelid Edge Detection: <Name>
 Bottom Eyelid Edge Detection: <Name>
Normalisation: <Name>
 Segmentation Mask: <Name>
Encoding: <Name>
Matching: <Name>

Number of records:<>
Time to process: <Time>
Average time per record: <Time> seconds
Iris code style: <Name>

```
<Name>
=====
Analysis data
  File name:    <Name>.idb
  Processing time:  <Time>

True Matches
  Count:  TM / Max (<>%)
  Hamming distance range:<> to <MaxTrueMatch>
  True matches with Hamming dist < MinFalseMatch: <>%

False Matches
  Count:  FM / Max (<>%)
  Hamming distance range:<MinFalseMatch> to < MaxFalseMatch >
  False matches with Hamming dist > MaxTrueMatch: <>%
```

The text file is separated into logical sections. On the first page, at the top, below the creation date and time, are the summary statistics. These are the headline results and information. The image and results dataset names are stated followed by a section containing the recognition results. This is followed by the speed results.

On the second page, the system info is included for future comparison, with details of the hardware, OS and MATLAB version used. This is followed by the information on the algorithms used. This will state which algorithm was selected for each stage of iris recognition.

The final page contains more detailed information regarding the analysis data and the true and false matches. The <Name> field is the name selected by the user in for the results dataset.

Appendix H Quick-Start User Manual

The purpose of the Iris GUI tool is as a generic platform to aid the development and evaluation of iris recognition algorithms. This quick-start manual is aimed at getting started with using Iris GUI as quickly as possible, for more detail, please see the technical user guide wiki on SourceForge.

H.1 Installation and Setup of the Generic Platform

Step One: Ensure that MATLAB is installed on your machine. All versions from 2007a are supported by Iris GUI.

Step Two: Download the Iris GUI zip file from either the MathWorks MATLAB Central web site, <http://www.mathworks.co.uk/matlabcentral/> or Source Forge, <http://sourceforge.net/>.

Step Three: Extract the contents of the downloaded Zip file.

In Microsoft Windows 7 this is achieved by right clicking on the file with the mouse and clicking ‘Extract All...’ with the left mouse button in the menu that opens, as shown in Figure 6.12.

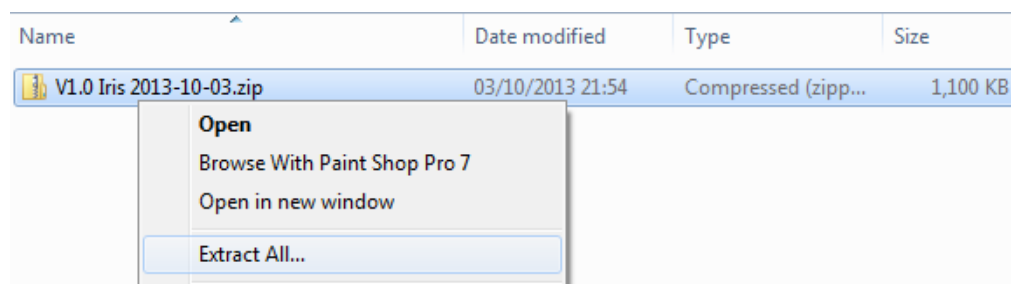


Figure 6.12 – Extracting the Iris GUI zip file using the mouse in Windows 7

Upon selecting the 'Extract All...' option Windows will display the zip file extraction dialog shown in Figure 6.13.

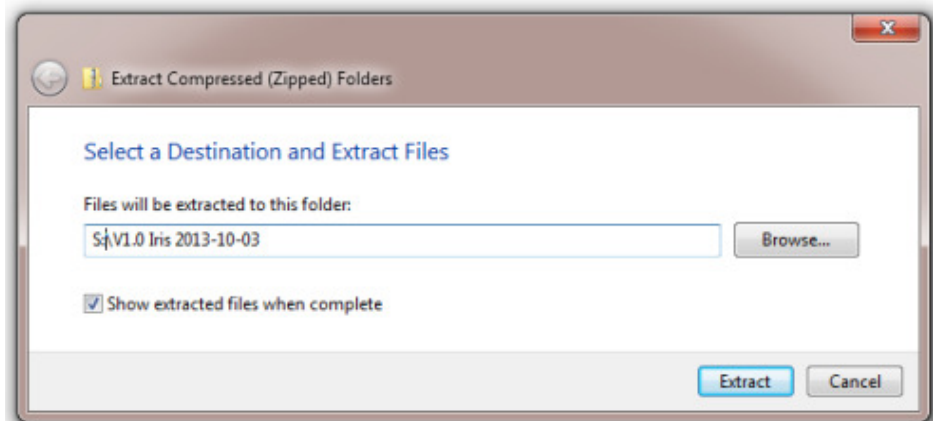


Figure 6.13 – The zip file extraction dialog window in Windows 7

Set the folder name in the zip extraction dialog to a location that is convenient. Shorter path names will simplify the development process. The folder created here will be referred to the 'Iris GUI home folder' in the following steps. An appropriate folder location and name would be:

C:\IrisGUI\Program\

Then left click the 'Extract' Button in the zip extraction dialog.

Step Four: Increase the Java heap space available to MATLAB.

For version's of MATLAB between version 6.0 (R12) and version 7.9 (2009b) this is achieved by adding a 'Java.opts' file to the \$MATLABROOT/bin/\$ARCH directory. On 64-bit Windows, running 64-bit MATLAB this translates to:

C:\Program Files\MATLAB\R2009a\bin\win64

A suitable 'Java.opts' file is provided with the generic platform in the Iris GUI home folder. Just copy this file to the correct MATLAB folder for your system. For more detailed information, including other versions of MATLAB please see:

<http://www.mathworks.com/matlabcentral/answers/92813-how-do-i-increase-the-heap-space-for-the-java-vm-in-matlab-6-0-r12-and-later-versions>

Step Five: Run MATLAB and add the Iris GUI home folder to the MATLAB path.

To add the Iris GUI home folder to the MATLAB path in MATLAB version 2009a, run MATLAB, then access the path dialog through the File menu, ‘Set Path...’, shown in Figure 6.14

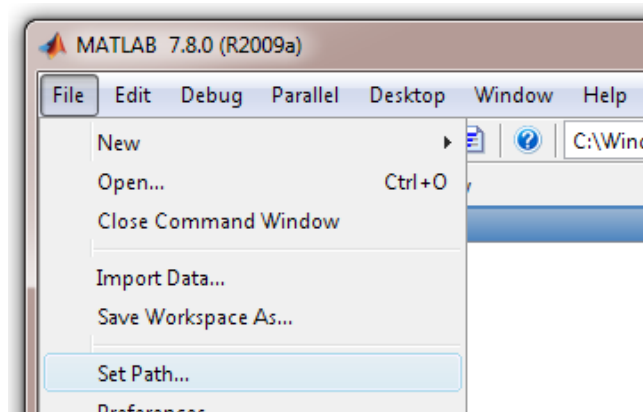


Figure 6.14 – The ‘Set Path...’ menu option in MATLAB

After selecting the ‘Set Path...’ menu option the ‘Set Path’ dialog box will be displayed. Clicking the ‘Add Folder’ button shown in Figure 6.15 will allow user to navigate to the Iris GUI home folder.

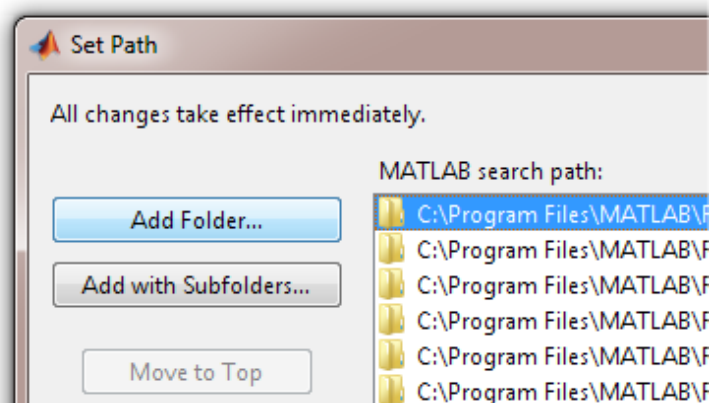


Figure 6.15 – The ‘Add Folder’ button in the ‘Set Path’ dialog box

Once the Iris GUI home folder has been added to the ‘MATLAB Search Path’ list, it should be moved to the top of the list by clicking the ‘Move to Top’ followed by the ‘Save’ buttons highlighted in Figure 6.16.

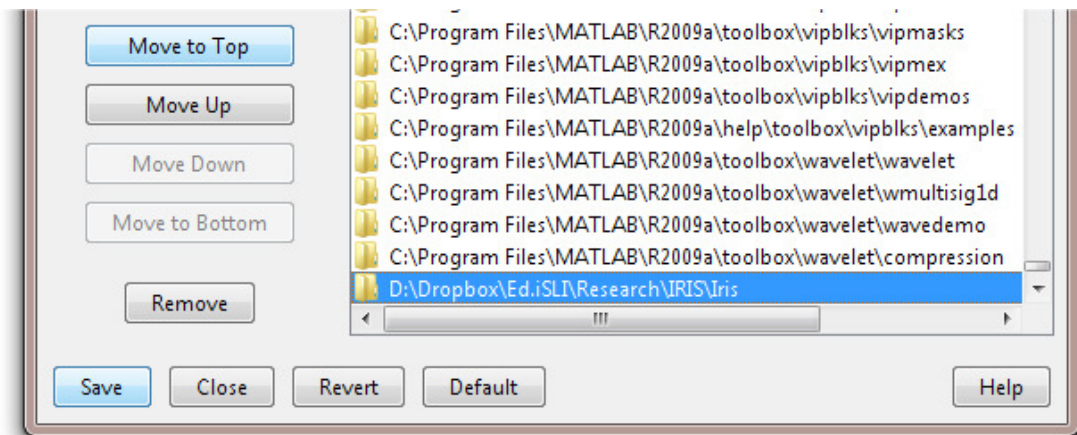


Figure 6.16 – The ‘Move to Top’ and ‘Save’ buttons in the ‘Set Path’ dialog box

Step Six: Create the iris image database and results database directories.

There are no restrictions on where these directories are placed, or what they are named, outside any limitations imposed by the host operating system on folder naming and placement. Though shorter paths may simplify development, a suitable example would be:

C:\IrisGUI\imageDB\

C:\IrisGUI\resultsDB\

This is also a good time to add the image datasets to be used. These are added to the image database directory, with each image dataset in its own folder under the ‘ImageDB’ directory, as shown in Figure 6.17.

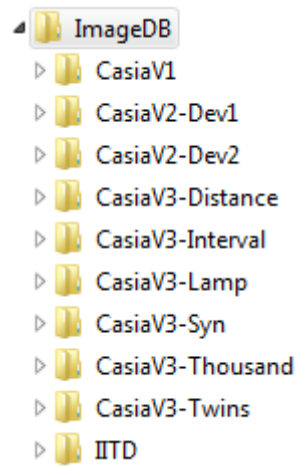


Figure 6.17 – An example image database

Step Seven: Run the generic platform.

At the MATLAB command prompt type 'Iris', without the quotes, and press enter. The generic platform will first request the location of the image database directory created in step six, as shown in Figure 6.18.

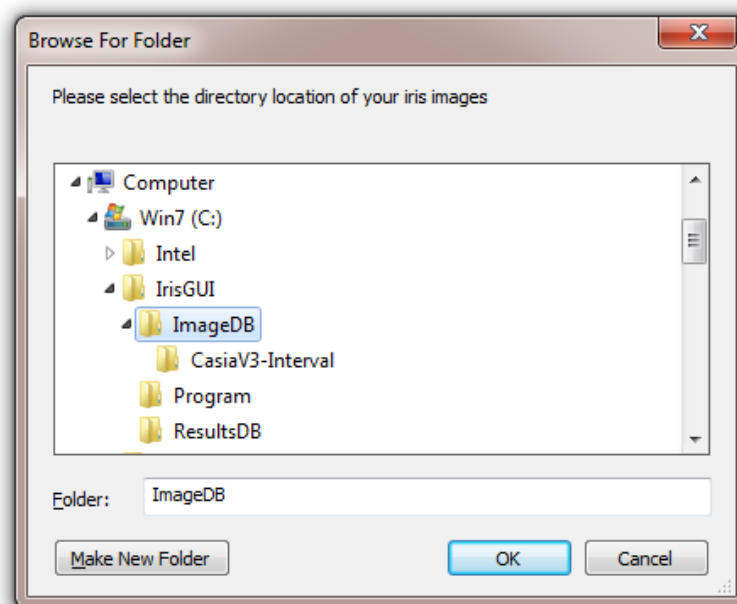


Figure 6.18 – The dialog box for selecting the image database location

After selecting the image database directory click 'OK' and the generic platform will then ask where to store its results information, as shown in Figure 6.19.

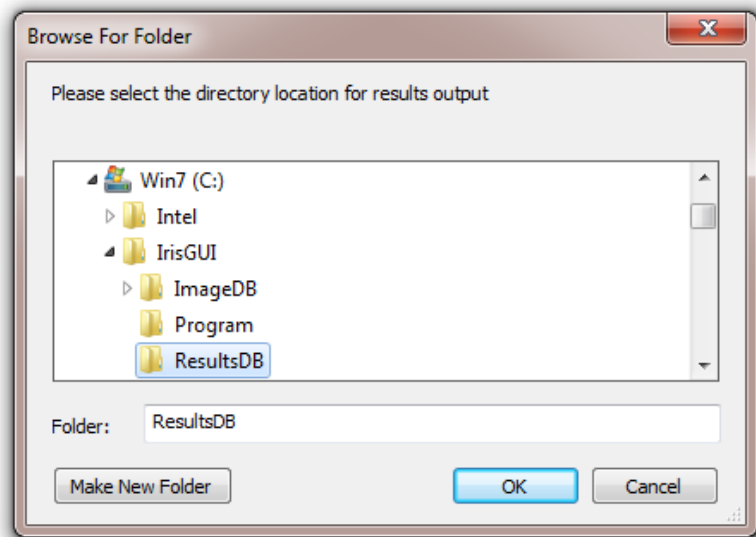


Figure 6.19 – The dialog box for selecting the result database location

Once the results database directory has been selected and the ‘OK’ button clicked, the generic platform GUI will load as shown in Figure 6.20.

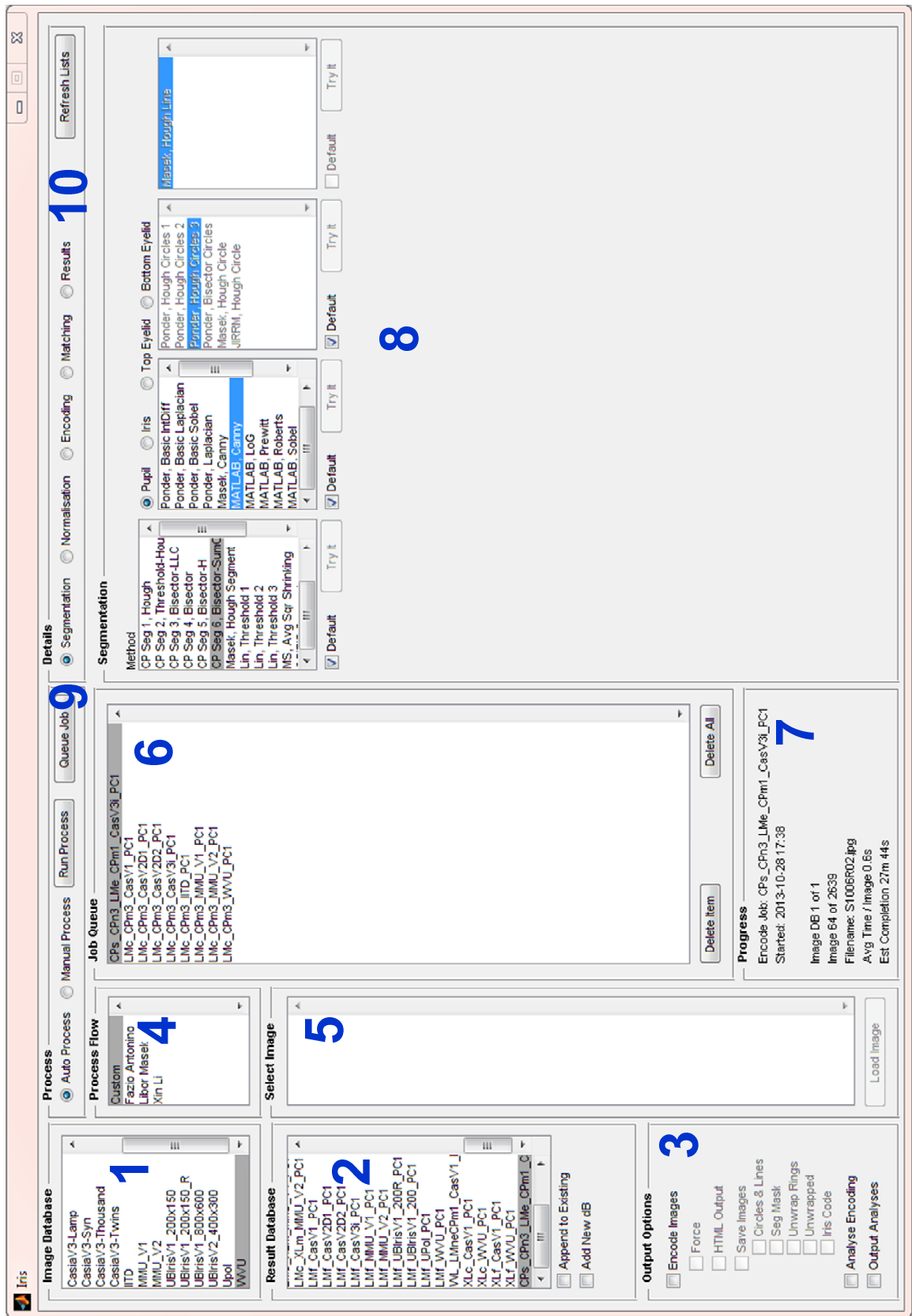


Figure 6.20 – The generic platform GUI in Automatic Mode

Table 6.10 provides more information on the areas marked with blue numbers in Figure 6.20

Table 6.10 – Description of the generic platform GUI regions from Figure 6.20


Image Region	Name	Description
1	Image Database	The available image datasets
2	Result Database	The available results datasets
3	Output Options	The available options for the currently selected iris recognition flow and results dataset combination
4	Process Flow	Selecting custom in this list enables the selection of segmentation , normalisation and encoding algorithms in area (8). Selecting another option will
5	Image List	The list all of the images in the currently selected image dataset
6	Job Queue	The list of jobs queued for processing
7	Progress	The progress of the currently active job
8	Algorithm Selection	This area is where the segmentation, normalisation and encoding can be selected when the ‘Custom’ process flow is selected in (4)
9	Process Selection	This area contains the toggle between automatic and manual modes as well as the ‘Queue Job’ button for adding jobs to the Job Queue and the ‘Run Process’ button for starting the processing of any queued jobs
10	Details Selection	This area changes the information in area (8) between segmentation, normalisation, encoding, matching and results display. The ‘Refresh lists’ button instructs the GUI to re-load all the modules, image and results datasets in the event that these have changed

H.2 Getting Started With Using the Generic Platform

Now the generic platform is setup and running it is possible to carry out iris recognition tasks. This section assumes that there is at least one iris image dataset installed. If no iris image dataset is currently available then there is a free subset of the WVU image dataset at http://www.csee.wvu.edu/~xinl/demo/nonideal_iris.html.

To get started:

1. Select an image dataset in area (1) in Figure 6.20 and wait for the image dataset to load
2. Change the generic platform from Automatic to Manual in area (9)
3. Select 'CP Seg 3' from the segmentation 'Method' list box in area (8)
4. Click the 'Try It' button below the segmentation 'Method' list box in area (8)

Area (8) will very quickly display the result of this segmentation operation as shown in Figure 6.21. The small  buttons in the top left corner of images will allow the images to be opened in a separate figure window.

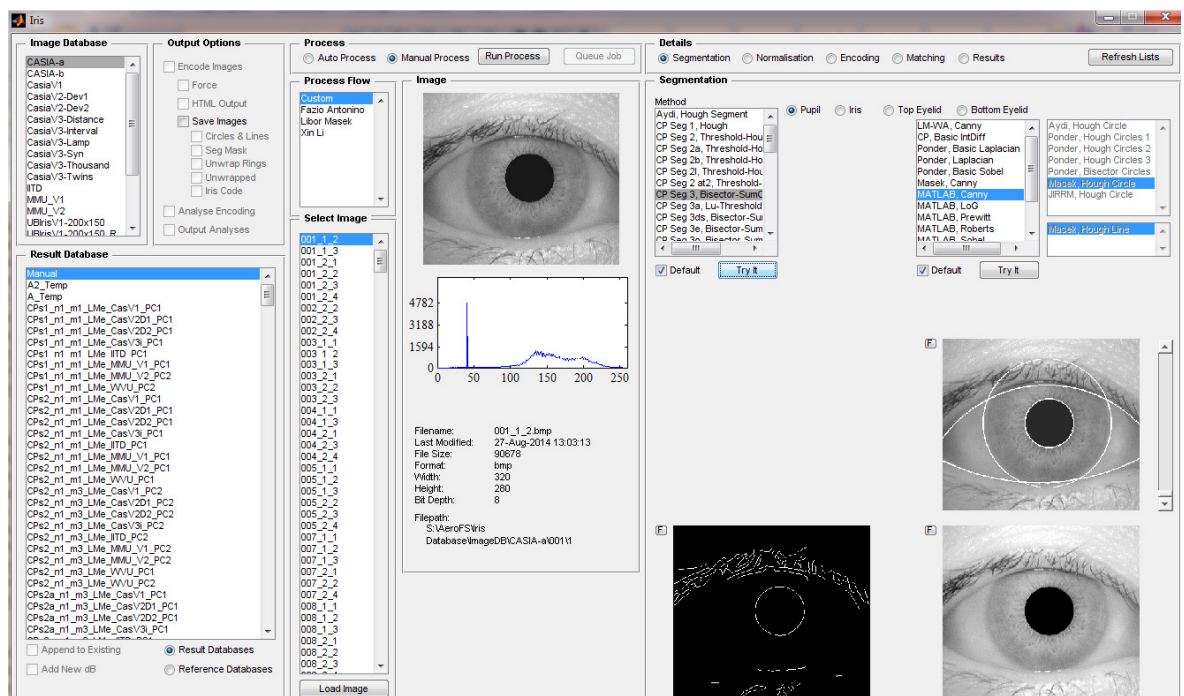


Figure 6.21 – The generic platform GUI in Manual Mode showing the results of a segmentation operation

5. Change area (8) to normalisation by selecting the 'Normalisation' option in area (10)
6. Select 'CP Norm 1' from the normalisation 'Method' list box in area (8)
7. Click the 'Try It' button below the normalisation 'Method' list box in area (8)

Area (8) will very quickly display the result of this normalisation operation as shown in Figure 6.22.

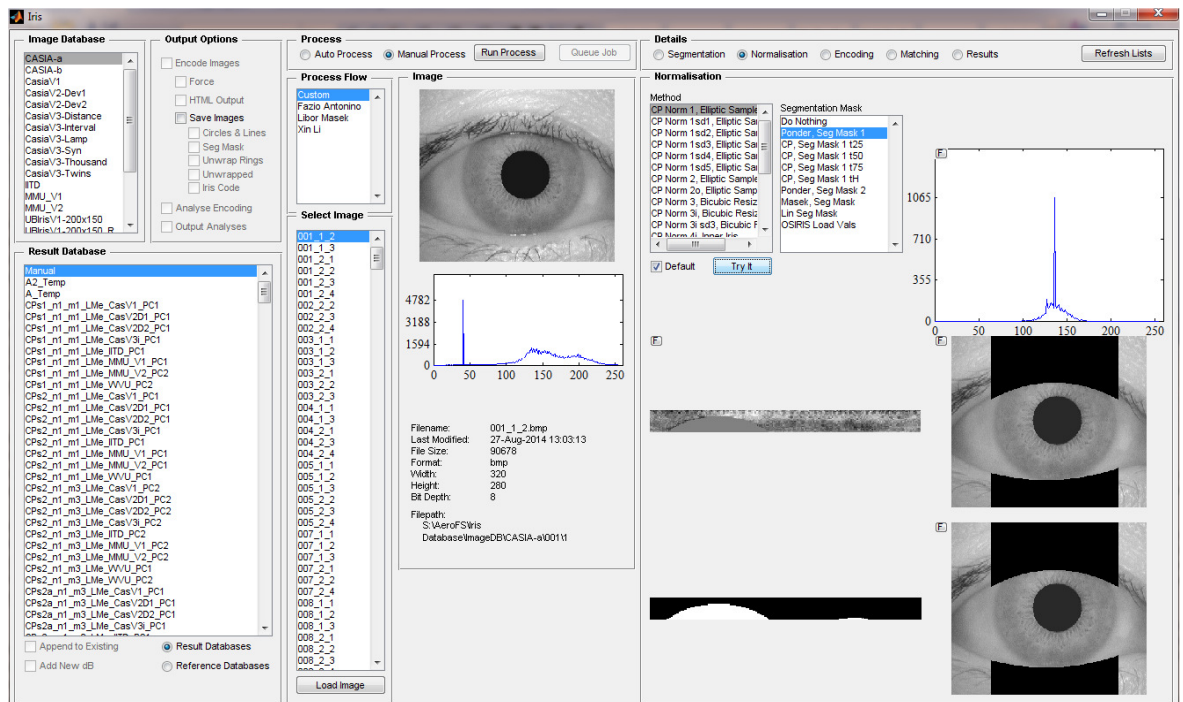


Figure 6.22 – The generic platform GUI in Manual Mode showing the results of a normalisation operation

8. Change area (8) to encoding by selecting the 'Encoding' option in area (10)
9. Select 'Masek, Encode' from the encoding 'Method' list box in area (8)
10. Click the 'Try It' button below the encoding 'Method' list box in area (8)

Area (8) will very quickly display the result of this encoding operation as shown in Figure 6.23.

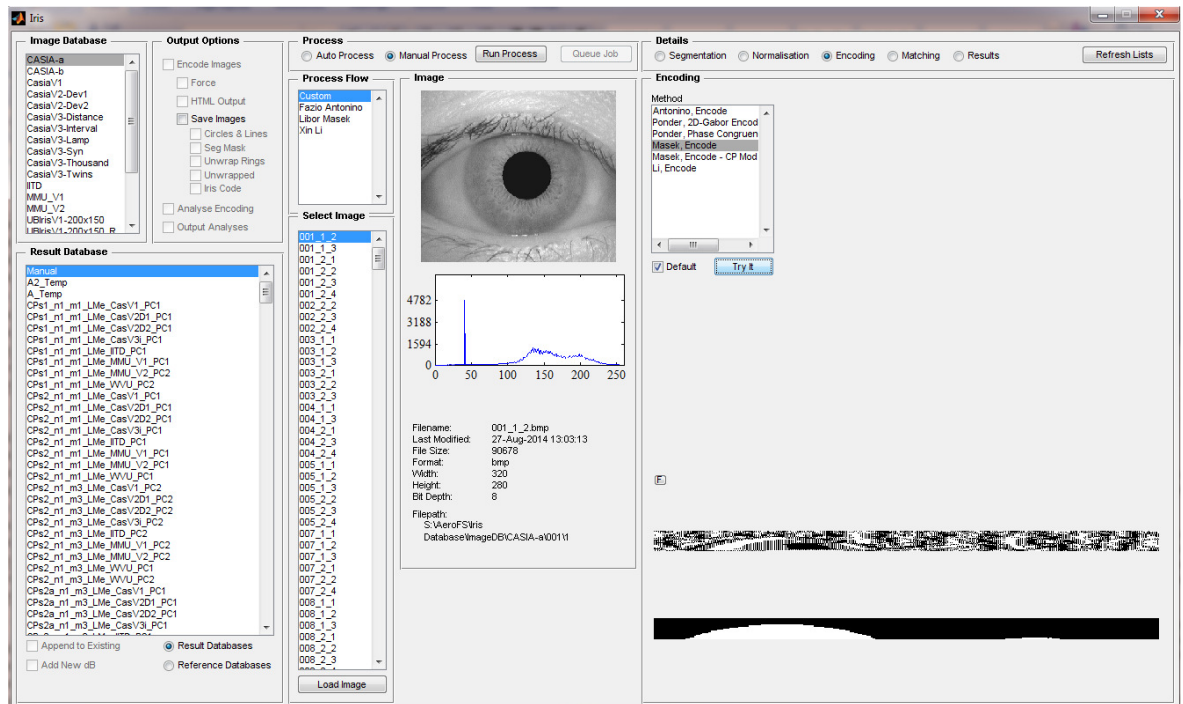


Figure 6.23 – The generic platform GUI in Manual Mode showing the results of an encoding operation

11. Change the generic platform from Manual to Automatic in area (9)
12. Check the 'Add New dB' check box at the bottom of area (2)
13. Type a name for the results dataset. A suitable name might be:
CPs3_n1_m1_LMe_DatasetName
14. Change area (8) to matching by selecting the 'Matching' option in area (10)
15. Select 'CP Match 1' from the encoding 'Method' list box in area (8)
16. In area (3), check the 'Encode Images', 'Save Images' 'Circles & Lines', 'Analyse Encoding' and 'Output Analyses' check boxes
17. Click the 'Run Processes' button

The encoding and analysis of the selected image dataset with the selected iris recognition algorithms will be started. The progress of this job will be shown in area (7), as shown in Figure 6.24.

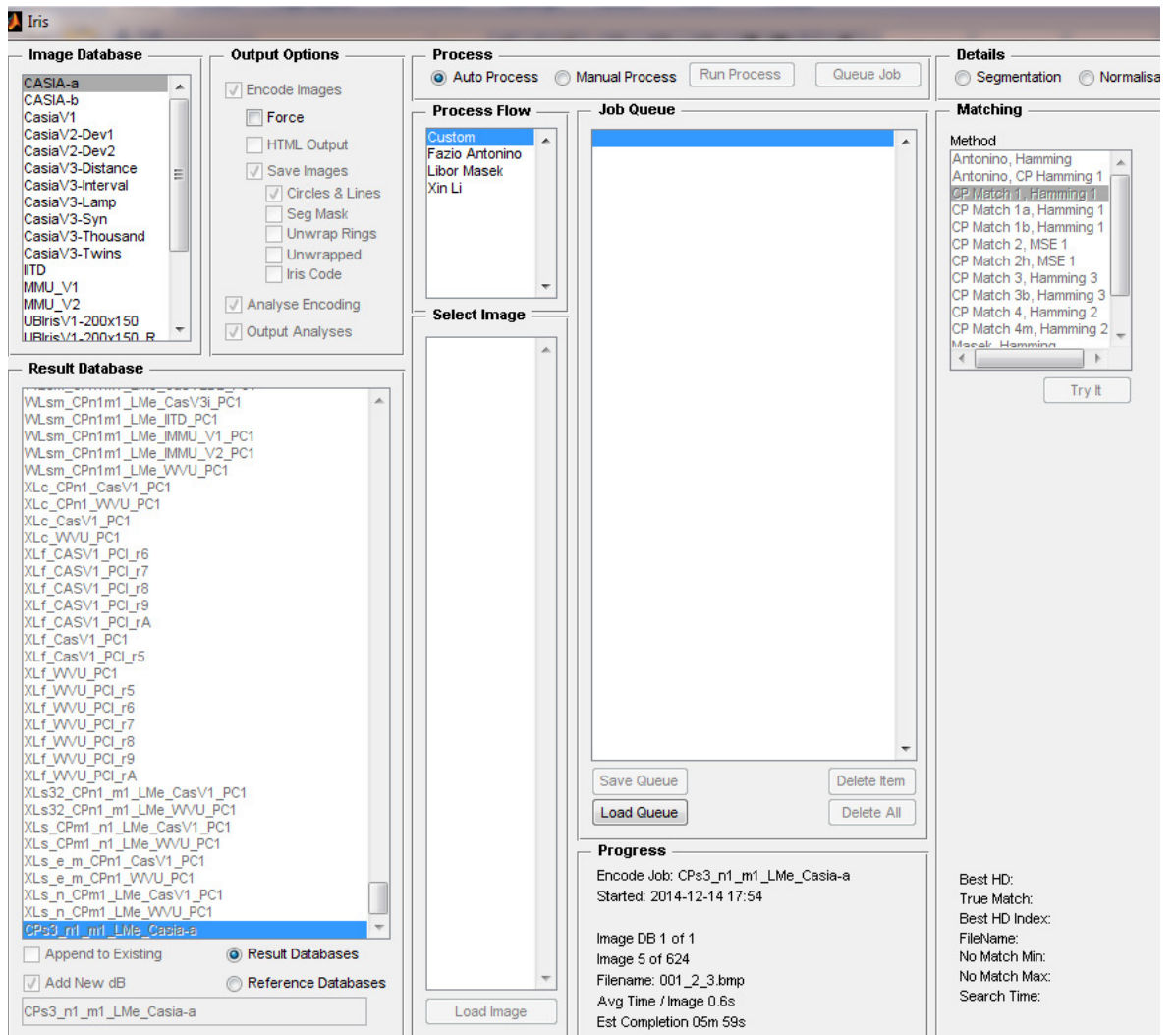


Figure 6.24 – The generic platform GUI in Automatic Mode processing the selected image dataset with the selected iris recognition algorithms

The results from this process will be output to the results database directory defined during setup in a subdirectory named the same as the results dataset name.

This concludes the Quick-Start guide. More detailed information is available in the technical user guide wiki on SourceForge.

7 Bibliography

- [1] J. Daugman, "High confidence personal identification by rapid video analysis of iris texture," in *IEEE International Carnahan Conference on Security Technology, Crime Countermeasures, Proceedings.*, Atlanta, GA, 1992.
- [2] J. Daugman, "High confidence visual recognition of persons by a test of statistical independence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1148-1161, 1993.
- [3] M. Nabti and A. Bouridane, "Recent Advances in Iris Recognition: A Multiscale Approach," in *Imaging for Forensics and Security*, Springer Science+Business Media LLC, 2009, pp. 49-77.
- [4] T. Tan, "Note on CASIA-IrisV1," Chinese Academy of Sciences' Institute of Automation, 03 2011. [Online]. Available: <http://biometrics.idealtest.org/>. [Accessed 04 2011].
- [5] T. Tan, "Note on CASIA-IrisV2," Chinese Academy of Sciences' Institute of Automation, 03 2011. [Online]. Available: <http://biometrics.idealtest.org/>. [Accessed 04 2011].
- [6] T. Tan, "Note on CASIA-IrisV3," Chinese Academy of Sciences; Institute of Automation, 03 2011. [Online]. Available: <http://biometrics.idealtest.org/>. [Accessed 04 2011].
- [7] T. Tan, "Note on CASIA-IrisV4," Chinese Academy of Sciences; Institute of Automation, 03 2011. [Online]. Available: <http://biometrics.idealtest.org/>. [Accessed 04 2011].

- [8] H. T. Ewe, MULTIMEDIA UNIVERSITY, 07 2010. [Online]. Available: <http://pesona.mmu.edu.my/~ccteo/index.htm>. [Accessed 04 2011].
- [9] H. P. Proença and L. A. Alexandre, "UBIRIS.v1," University of Beira Interior, 02 2011. [Online]. Available: <http://iris.di.ubi.pt/ubiris1.html>. [Accessed 04 2011].
- [1 H. P. Proença and L. A. Alexandre, "UBIRIS.v2," University of Beira Interior, 02 0] 2011. [Online]. Available: <http://iris.di.ubi.pt/ubiris2.html>. [Accessed 04 2011].
- [1 West Virginia University, "Biometric Dataset Collections," West Virginia University, 1] 2005. [Online]. Available: http://www.citer.wvu.edu/biometric_dataset_collections. [Accessed 03 2011].
- [1 X. Li, "Modeling Intra-class Variation for Nonideal Iris Recognition," in *ICB2006*, 2] Hong Kong, 2006.
- [1 Indian Institute of Technology Delhi, "IIT Delhi Iris Database version 1.0," IIT Delhi, 3] 2007. [Online]. Available: http://www4.comp.polyu.edu.hk/~csajaykr/IITD/Database_Iris.htm. [Accessed 05 2012].
- [1 A. Kumar and A. Passi, "Comparison and Combination of Iris Matchers for Reliable 4] Personal Authentication," *Pattern Recognition*, vol. 43, pp. 1016-1026, Mar. 2010.
- [1 D. M and L. Machala, "Iris Database," Palacky University in Olomouc, 2009. [Online]. 5] Available: <http://phoenix.inf.upol.cz/iris/>. [Accessed 05 2011].
- [1 M. Dobeš, J. Martinek, D. Skoupi, Z. Dobešová and J. Pospíšil, "Human eye 6] localization using the modified Hough transform," *Optik*, vol. 117, no. 10, pp. 468-473, 2006.
- [1 National Institute of Standards and Technology (NIST), "ICE2005 Database," NIST, 7] 2005. [Online]. Available: <http://iris.nist.gov/ice/>. [Accessed 03 2011].

- [1 Unique Identification Authority of India, Planning Commission, Government of India, 8] “Aadhaar Services - Resident Portal,” Government of India, 2012. [Online]. Available: http://resident.uidai.net.in/en_GB/aadhaar-services. [Accessed 04 2012].
- [1 The Child Project, “The Child Project - Home,” The Child Project, 25 09 2007. 9] [Online]. Available: <http://www.thechildproject.org/>. [Accessed 02 08 2010].
- [2 H. Proença and L. Alexandre, “Noisy Iris Challenge Evaluation (NICE) - Part I,” 0] SOCIA Lab. – Soft Computing and Image Analysis Group, 2009. [Online]. Available: <http://nice1.di.ubi.pt/>. [Accessed 2011].
- [2 H. Proença and L. Alexandre, “Noisy Iris Challenge Evaluation (NICE) - Part II,” 1] SOCIA Lab. – Soft Computing and Image Analysis Group, 2010. [Online]. Available: <http://nice2.di.ubi.pt/>. [Accessed 2011].
- [2 H. Proença, “Iris Recognition: On Segmentation of Degraded Images Acquired in the 2] Visible Wavelength,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1502-1516, 08 2010.
- [2 H. Proença, “Quality Assessment of Degraded Iris Images Acquired in the Visible 3] Wavelength,” *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 1, pp. 82-95, 02 2011.
- [2 J. Matey, O. Naroditsky, K. Hanna, R. Kolczynski, D. LoIacono, S. Mangru, M. 4] Tinker, T. Zappia and W. Zhao, “Iris on the Move: Acquisition of Images for Iris Recognition in Less Constrained Environments,” *Proceedings of the IEEE*, vol. 94, no. 11, pp. 1936 - 1947, 2006.
- [2 H. Proença, S. Filipe, R. Santos, J. Oliveira and L. Alexandre, “The UBIRIS.v2: A 5] Database of Visible Wavelength Iris Images Captured On-the-Move and At-a-Distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1529-1535, 08 2010.
- [2 J. Daugman, “Probing the Uniqueness and Randomness of IrisCodes: Results From 200

- 6] Billion Iris Pair Comparisons,” *Proceedings of the IEEE*, vol. 94, no. 11, pp. 1927-1935, 11 2006.
- [2 X. Li, “Xin Li’s Homepage,” West Virginia University, 2005. [Online]. Available:
7] http://www.csee.wvu.edu/~xinl/demo/nonideal_iris.html. [Accessed 01 2011].
- [2 C. Rathgeb, A. Uhl and P. Wild, “Iris-Biometric Comparators: Minimizing Trade-Offs
8] Costs between Computational Performance and Recognition Accuracy,” in *Proceedings of the 4th International Conference on Imaging for Crime Detection and Prevention*, London, 2011.
- [2 L. Masek, “Open Source Iris Recognition Implementation,” The University of Western
9] Australia, 07 2010. [Online]. Available:
<http://www.csse.uwa.edu.au/~pk/studentprojects/libor/>. [Accessed 02 08 2010].
- [3 W. T. Lin, C. C. Liu and S. Y. Chen, “Fast and robust iris recognition,” *The Imaging
0] Science Journal*, vol. 57, no. 3, pp. 128-139, 06 2009.
- [3 L. Masek, “Recognition of Human Iris Patterns,” 2003. [Online]. Available:
1] <http://www.csse.uwa.edu.au/~pk/studentprojects/libor/>. [Accessed 2010 08 02].
- [3 T. Trebaol and M. Savvides, “Fractal encoding of low resolution iris imagery for
2] improved matching,” in *37th IEEE Applied Imagery Pattern Recognition Workshop*, Washington DC, 2008.
- [3 A. Murugan and G. Savithiri, “Feature extraction on half iris for personal
3] identification,” in *Conference on Signal and Image Processing (ICSIP), 2010 International*, Chennai, 2010.
- [3 J. Zuo, N. Kalka and N. Schmid, “A Robust IRIS Segmentation Procedure for
4] Unconstrained Subject Presentation,” in *Special Session on Research at the Biometric Consortium Conference, 2006 Biometrics Symposium*, Baltimore, MD, 2006.
- [3 J. Zuo and N. Schmid, “On a Methodology for Robust Segmentation of Nonideal Iris

- 5] Images,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 40, no. 3, pp. 703-718, 06 2010.
- [3 K. W. Bowyer, K. Hollingsworth and P. J. Flynn, “Image Understanding for Iris
6] Biometrics: A Survey,” *Computer Vision and Image Understanding*, vol. 110, no. 2, pp. 281-307, 2008.
- [3 X. Liu, K. Bowyer and P. Flynn, “Experiments with an improved iris segmentation
7] algorithm,” in *IEEE Workshop on Automatic Identification Advanced Technologies*, Buffalo, NY, USA, 2005.
- [3 F. G. Antonino, “Iris Recognition | Sourcforge.net,” 04 2011. [Online]. Available:
8] <http://sourceforge.net/projects/iris-ict/>. [Accessed 09 2010].
- [3 G. Sutra and B. O. N. Dorizzi, “Iris_Osiris_v4.1,” Telecom Sud Paris, 2012. [Online].
9] Available: <http://share.int-evry.fr/svnview-eph/>. [Accessed 08 2012].
- [4 cs02rm0, “JIRRM | SourceForge.net,” Open-source, 19 11 2004. [Online]. Available:
0] <http://sourceforge.net/projects/jirrm/>. [Accessed 11 2010].
- [4 R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves
1] in pictures,” *Communications of the ACM*, vol. 15, no. 1, pp. 11-15, 01 1972.
- [4 L. Rosa, “High Performance Iris Recognition Based on DCT,” 2007. [Online].
2] Available: <http://www.advancedsourcecode.com/dctiris.asp>. [Accessed 11 2010].
- [4 L. Rosa, “Iris Recognition System,” 2007. [Online]. Available:
3] <http://www.advancedsourcecode.com/iris.asp>. [Accessed 11 2010].
- [4 GruSoft, “GIRIST,” GruSoft, 2009. [Online]. Available:
4] <http://www.grusoft.com/girist.htm>. [Accessed 11 2010].
- [4 J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on*

- 5] *Pattern Analysis and Machine Intelligence*, Vols. PAMI-8, no. 6, pp. 679-698, 11 1986.
- [4 N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62-66, 01 1979.
- [4 R. Wildes, J. Asmuth, G. Green, S. Hsu, R. Kolczynski, J. Matey and S. McBride, "A
7] system for automated iris recognition," in *Proceedings of the Second IEEE Workshop on Applications of Computer Vision*, Sarasota, FL, 1994.
- [4 F. H. Adler, *Physiology of the Eye: Clinical Application*, Bethesda MD: Mosby, 1953.
8]
- [4 R. G. Johnson, "Can iris patterns be used to identify people," Los Alamos National
9] Laboratory, CA, Chemical and Laser Sciences Division, Rep. LA-12331-PR, 1991.
- [5 J. Daugman and C. Downing, "Recognizing iris texture by phase demodulation," in
0] *IEE Colloquium on Image Processing for Biometric Measurement*, London, 1994.
- [5 R. Wildes, "Iris recognition: an emerging biometric technology," *Proceedings of the*
1] *IEEE*, vol. 85, no. 9, pp. 1348 - 1363 , Sep 1997.
- [5 Y.-H. Li and M. Savvides, "Iris Recognition," in *Biometrics: Theory, Methods and*
2] *Applications*, N. V. Boulgouris, K. N. Plataniotis and E. Micheli-Tzanakou, Eds.,
Wiley-IEEE Press, 2009, pp. 315-338.
- [5 OKI, "Oki Electric Begins Sales of "IRISPASS®-h", an Iris Authentication Unit with
3] USB Support, Available for PC Connection," OKI, 28 11 2001. [Online]. Available:
<http://www.oki.com/en/press/2001/z0190e.html>. [Accessed 05 2011].
- [5 Iris ID, "Iris ID - Iris Recognition Technology," Iris ID / LG, 1999. [Online].
4] Available: <http://irisid.com/>. [Accessed 05 2011].
- [5 Y. Du, E. Arslanturk, Z. Zhou and C. Belcher, "Video-Based Noncooperative Iris
Image Segmentation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B:*

- 5] *Cybernetics*, vol. 41, no. 1, pp. 64-74, 02 2011.
- [5 R. Jillela, A. Ross and P. Flynn, "Information fusion in low-resolution iris videos using
6] Principal Components Transform," in *IEEE Workshop on Applications of Computer
7] Vision (WACV)*, Kona, HI, 2011.
- [5 M. Carneiro, A. Veiga, S. Silva, E. Flores and G. Carrijo, "Analysing the performance
7] of the algorithms used to localize the iris region in eye images submitted to severely
compressed images," in *IEEE International Symposium on Intelligent Signal
Processing, 2009. WISP 2009*, Budapest, 2009.
- [5 J. Daugman and C. Downing, "Effect of Severe Image Compression on Iris
8] Recognition Performance," *IEEE Transactions on Information Forensics and Security*,
vol. 3, no. 1, pp. 52-61, 02 2008.
- [5 S. Rakshit and D. Monro, "An Evaluation of Image Sampling and Compression for
9] Human Iris Recognition," *IEEE Transactions on Information Forensics and Security*,
vol. 2, no. 3, pp. 605-612, 09 2007.
- [6 H.-q. Lu, "Defining Iris Boundary Detail Method for Iris Localization," in *IEEE
0] International Symposium on Knowledge Acquisition and Modeling Workshop*, Wuhan,
2008.
- [6 H. J. Koh, W. J. Lee and M. G. Chun, "A Multimodal Iris Recognition Using Gabor
1] Transform and Contourlet," in *ICSPCS 2008. 2nd International Conference on Signal
Processing and Communication Systems*, Gold Coast, 2008.
- [6 M. Shamsi, P. Saad, S. Ibrahim and A. Kenari, "Fast Algorithm for Iris Localization
2] Using Daugman Circular Integro Differential Operator," in *International Conference of
Soft Computing and Pattern Recognition*, Malacca, 2009.
- [6 E. Dhir and M. Dutta, "New Method of Iris Recognition Based on J.Daugman's
3] Principle," in *International Conference on Emerging Trends in Engineering and*

Technology, Nagpur, 2009.

- [6 Z. Osman, "Iris Recognition Using Phase Congruency," in *International Conference on 4] Computer Modelling and Simulation*, Cambridge, 2011.
- [6 S. Ziauddin and M. Dailey, "Iris recognition performance enhancement using weighted 5] majority voting," in *15th IEEE International Conference on Image Processing. (ICIP)*, San Diego, CA , 2008.
- [6 Sony, "Support for DSC-F717 | Sony," Sony, 2002. [Online]. Available: 6] <http://www.sony.co.uk/support/en/product/dsc-f717>. [Accessed 05 2011].
- [6 JIRIS, "JPC1000 Iris Recognition Camera (Sensor) Module," JIRIS, 03 2006. [Online]. 7] Available: <http://jiristech.en.ecplaza.net/jpc1000-iris-recognition-camera-sensor--82633-205986.html>. [Accessed 06 2011].
- [6 TOPCON, "TOPCON," TOPCON, 1997. [Online]. Available: [http://www.topcon-8\] medical.eu/eu/](http://www.topcon-8] medical.eu/eu/). [Accessed 06 2011].
- [6 Sony, "DXC-990P (DXC990P): Product Overview," Sony, 2001. [Online]. Available: 9] <http://www.sony.co.uk/pro/product/medical-cameras/dxc-990p/overview/>. [Accessed 05 2011].
- [7 "Intensity Transformations and Spatial Filtering," in *Digital Image Processing Using 0] MATLAB*, 2nd ed., New Jersey, Gatesmark Publishing, 2009, pp. 80-163.
- [7 Y. Hong, H. Wang and S. Kwong, "Image thresholding based on Random spatial 1] sampling and Majority voting," in *International Conference on Machine Learning and Cybernetics*, Qingdao, 2010.
- [7 D.-U. Lee, R. Cheung and J. Villasenor, "A Flexible Architecture for Precise Gamma 2] Correction," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, no. 4, pp. 474-478, April 2007.

- [7 M. Doustar and H. Hassanpour, “A locally-adaptive approach for image gamma
3] correction,” in *International Conference on Information Sciences Signal Processing
and their Applications*, Kuala Lumpur, 2010.
- [7 W. Gao, X. Zhang, L. Yang and H. Liu, “An improved Sobel edge detection,” in *IEEE
4] International Conference on Computer Science and Information Technology*, Chengdu,
2010.
- [7 Y. Hao, L. Changshun and P. Lei, “An improved method of image edge detection
5] based on wavelet transform,” in *IEEE International Conference on Computer Science
and Automation Engineering*, Shanghai, 2011.
- [7 J. Daugman, “How iris recognition works,” *IEEE Transactions on Circuits and Systems
6] for Video Technology*, vol. 14, no. 1, pp. 21-30, 01 2004.
- [7 J. Daugman, “How iris recognition works,” in *International Conference on Image
7] Processing.*, 2002.
- [7 J. Daugman, “New Methods in Iris Recognition,” *IEEE Transactions on Systems, Man,
8] and Cybernetics, Part B: Cybernetics*, vol. 37, no. 5, pp. 1167-1175, 10 2007.
- [7 M. Shamsi and A. Kenari, “Iris boundary detection using an ellipse integro differential
9] method,” in *2nd International eConference on Computer and Knowledge Engineering
(ICCKE)*, Mashhad, 2012.
- [8 Y. Xie and Q. Ji, “A new efficient ellipse detection method,” in *16th International
0] Conference on Pattern Recognition*, 2002.
- [8 J. Daugman, “High confidence recognition of persons by iris patterns,” in *IEEE 35th
1] International Carnahan Conference on Security Technology*, London, 2001.
- [8 J. Huang, Y. Wang, T. Tan and J. Cui, “A new iris segmentation method for
2] recognition,” in *International Conference on Pattern Recognition, 2004*, 2004.

- [8 L. Birgale and M. Kokare, "Iris Recognition Without Iris Normalization," *Journal of Computer Science*, vol. 6, no. 9, pp. 1042-1047, 2010.
- [8 I. Tomeo-Reyes, J. Liu-Jimenez, I. Rubio-Polo, J. Redondo-Justo and R. Sanchez-Reillo, "Input images in iris recognition systems: A case study," in *Systems Conference (SysCon), 2011 IEEE International*, Montreal, QC, 2011.
- [8 S. McCloskey, W. Au and J. Jelinek, "Iris Capture from Moving Subjects Using a Fluttering Shutter," in *IEEE International Conference on Biometrics: Theory Applications and Systems*, Washington, DC, 2010.
- [8 H. Gu, S. Qiao and C. Yang, "An efficient iris localization algorithm based on standard deviations," in *International Workshop on Open-Source Software for Scientific Computation (OSSC), 2011*, Beijing , 2011.
- [8 W. Aydi, N. Masmoudi and L. Kamoun, " Improved Masek approach for iris localization," in *International Conference on Microelectronics (ICM), 2011*, Hammamet, 2011.
- [8 S. Emerich, E. Lupu and R. Arsinte, "A new approach to iris recognition," in *10th International Symposium on Signals, Circuits and Systems (ISSCS), 2011*, Iasi, 2011.
- [8 K. Hollingsworth, K. Bowyer and P. Flynn, "Improved Iris Recognition through Fusion of Hamming Distance and Fragile Bit Distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 12, pp. 2465 - 2476 , 2011.
- [9 W. W. Boles, "A Wavelet Transform Based Technique For The Recognition Of The Human Iris," in *International Symposium on Signal Processing and its Applications*, Brisbane, Australia, 1996.
- [9 J. Daugman, "Face and gesture recognition: overview," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 675-676, 07 1997.
- [9 J. Daugman, "Biometric decision landscapes," University of Cambridge, Cambridge,

2] 2000.

[9 G. Strang, "Wavelets," *American Scientist*, vol. 82, pp. 250-255, 04 1994.

3]

[9 D. Gabor, "Theory of communication," *Journal of the Institution of Electrical*
4] *Engineers - Part I: General*, vol. 94, no. 73, p. 58, 01 1947.

[9 D. Gabor, "Theory of communication. Part 1: The analysis of information," *Journal of*
5] *the Institution of Electrical Engineers - Part III: Radio and Communication*
Engineering, vol. 93, no. 26, pp. 429-441, 11 1946.

[9 D. Gabor, "Theory of communication. Part 2: The analysis of hearing," *Journal of the*
6] *Institution of Electrical Engineers - Part III: Radio and Communication Engineering*,
vol. 93, no. 26, pp. 442-445, 11 1946.

[9 D. Gabor, "Theory of communication. Part 3: Frequency compression and expansion,"
7] *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication*
Engineering, vol. 93, no. 26, pp. 445-457, 11 1946.

[9 R. A. King and T. C. Phipps, "Shannon, TESPAP and Approximation Strategies,"
8] *Computers & Security*, vol. 18, no. 5, pp. 445-453, 1999.

[9 K. Hollingsworth, K. Bowyer and P. Flynn, "The Best Bits in an Iris Code," *IEEE*
9] *Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 6, pp. 964 -
973 , 2009.

[1 R. Hamming, "Error detecting and error correcting codes," *The Bell System Technical*
00 *Journal*, vol. 29, no. 2, pp. 147-160, 04 1950.

]

[1 C. Rathgeb and A. Uhl, "Bit-Reliability-driven Template Matching in Iris
01 Recognition," in *Proc. 4th Pacific-Rim Symposium on Image and Video Technology*

] (*PSIVT'10*), Singapore, 2010.

[1 C. Rathgeb, A. Uhl and P. Wild, “Shifting Score Fusion: On Exploiting Shifting
02 Variation in Iris Recognition,” in *Proc. 26th ACM Symposium On Applied Computing*
] (*SAC'11*), Tai Chung, 2011.

[1 G. I. Davida, Y. Frankel and B. Matt, “On enabling secure applications through off-line
03 biometric identification,” in *IEEE Symposium on Security and Privacy, 1998.*
] *Proceedings.*, Oakland, CA , 1998.

[1 S. Yang and I. Verbauwhede, “Secure IRIS Verification,” in *IEEE International*
04 *Conference on Acoustics, Speech and Signal Processing. ICASSP*, Honolulu, HI, 2007.
]

[1 N. Boddeti and B. Kumar, “Extended Depth of Field Iris Recognition with Correlation
05 Filters,” in *2nd IEEE International Conference on Biometrics: Theory, Applications*
] *and Systems, 2008. (BTAS)*, Arlington, VA, 2008.

[1 E. Krichen, S. Garcia-Salicetti and B. Dorizzi, “A New Phase-Correlation-Based Iris
06 Matching for Degraded Images,” *IEEE Transactions on Systems, Man, and*
] *Cybernetics, Part B: Cybernetics*, vol. 39, no. 4, pp. 924 - 934, 2009.

[1 K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi and H. Nakajima, “An Effective Approach
07 for Iris Recognition Using Phase-Based Image Matching,” *IEEE Transactions on*
] *Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1741 - 1756, 2008.

[1 B. Patil and S. Subbaraman, “Human iris pattern recognition using phase components
08 of image,” in *International Conference on Industrial and Information Systems (ICIIS)*,
] Sri Lanka, 2009.

[1 M. Zhang, Z. Sun and T. Tan, “Perturbation-enhanced feature correlation filter for
09 robust iris recognition,” *The IEEE Biometrics Compendium*, vol. 1, no. 1, pp. 37 - 45,
] 2012.

[1 G. Sutra, B. Dorizzi and S. Garcia-Salicetti, "OSIRIS," BioSecure, 04 2012. [Online].
10 Available: <http://share.int-evry.fr/svnview-eph/>. [Accessed 08 2012].

]

[1 GruSoft, "GIRIST QUICK START USER GUIDE," GruSoft, Beijing, 2009.

11

]

[1 C. Ponder, K. Benkrid and A. Benkrid, "A generic computer platform for efficient iris
12 recognition," in *5th IAPR International Conference on Biometrics (ICB)*, New Delhi,
] 2012.

[1 H. Schildt, C++ The Complete Reference, 2nd Edition, Berkeley: Osbourne McGraw-
13 Hill, 1995.

]

[1 Tiobe Software, "Tiobe SOFTWARE Tiobe Index," Tiobe SOFTWARE, 07 2014.
14 [Online]. Available:

] <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. [Accessed 25 07
2014].

[1 H. Schildt, Java, A Beginner's Guide, 5th Edition, McGraw-Hill, 2012.

15

]

[1 V. Getov, Q. Lu, M. Thomas and M. Williams, "Message-passing computing with
16 Java: performance evaluation and comparisons," in *Ninth Euromicro Workshop on*
] *Parallel and Distributed Processing*, Mantova, 2001.

[1 N. Faria, R. Silva and J. Sobral, "Impact of Data Structure Layout on Performance," in
17 *21st Euromicro International Conference on Parallel, Distributed and Network-Based*
] *Processing (PDP)*, , Belfast, 2013.

[1 H. Alhussian, N. Zakaria, F. Hussin and H. Bahbouh, "A review of the current status of

18 the Java programming on embedded real-time systems,” in *International Conference*
] *on Computer & Information Science (ICCIS)*, Kuala Lumpur, 2012.

[1 C. Moler, “The Origins of MATLAB,” Mathworks, 12 2004. [Online]. Available:
19 <http://www.mathworks.co.uk/company/newsletters/articles/the-origins-of-matlab.html>.
] [Accessed 08 2010].

[1 N. Sharma and G. K. Matthias, “A Comparative Evaluation of MATLAB, OCTAVE,
20 FREEMAT and SCILAB for Research and Teaching,” 2010. [Online]. Available:
] <http://userpages.umbc.edu/~gobbert/papers/SharmaGobbertTR2010.pdf>. [Accessed
2010].

[1 GNU Octave, “FAQ - Octave,” GNU Octave, 14 06 2014. [Online]. Available:
21 http://wiki.octave.org/FAQ#How_is_Octave_different_from_Matlab.3F. [Accessed 21
] 07 2014].

[1 The Eclipse Foundation, “Eclipse CDT,” The Eclipse Foundation, 2010. [Online].
22 Available: <http://www.eclipse.org/cdt/>. [Accessed 10 2010].
]

[1 Code::Blocks, “Code::Blocks,” Code::Blocks, 2010. [Online]. Available:
23 <http://www.codeblocks.org/>. [Accessed 10 2010].
]

[1 SmartKoders, “CodeLite,” SmartKoders, 2010. [Online]. Available: <http://codelite.org/>.
24 [Accessed 10 2010].
]

[1 wxWidgets, “wxWidgets Cross-Platform GUI,” wxWidgets, 2014. [Online]. Available:
25 <http://www.wxwidgets.org/>. [Accessed 25 07 2014].
]

[1 OpenCV, “OpenCV,” OpenCV, 2010. [Online]. Available: <http://opencv.org>.
26

] [Accessed 10 2010].

[1 Python Software Foundation, "General Python FAQ," Python Software Foundation, 03 27 06 2014. [Online]. Available: <https://docs.python.org/3/faq/general.htm>. [Accessed 25 07 2014].

[1 M. Lutz, *Learning Python 5th Edition*, O'Reilly, 2013. 28]

[1 Numpy developers, "Numpy," Numpy developers, 2013. [Online]. Available: 29 <http://www.numpy.org/>. [Accessed 25 07 2014].]

[1 SciPy developers, "SciPy," SciPy developers, 2014. [Online]. Available: 30 <http://www.scipy.org/>. [Accessed 25 07 2014].]

[1 The matplotlib development team, "matplotlib: python plotting," The matplotlib 31 development team, 10 10 2013. [Online]. Available: <http://matplotlib.org/>. [Accessed] 25 07 2014].

[1 JetBrains s.r.o., "Python IDE & Django IDE for Web developers," JetBrains s.r.o., 32 2014. [Online]. Available: <http://www.jetbrains.com/pycharm/>. [Accessed 25 07 2014].]

[1 Wingware, "Python IDE for Python Developers," Wingware, 2014. [Online]. 33 Available: <http://www.wingware.com/>. [Accessed 25 07 2014].]

[1 Y. Du, R. Ives, D. M. Etter and T. Welch, "A new approach to iris pattern recognition," 34 in *In SPIE European Symposium on Optics/Photonics in Defence and Security*, 2004.]

- [1 C. C. Teo and H. T. Ewe, “An Efficient One-Dimensional Fractal Analysis for Iris
35 Recognition,” in *Proceedings of the 13th WSCG International Conference on*
] *Computer Graphics, Visualization and Computer Vision*, Central Europe, 2005.
- [1 M. Vatsa, R. Singh and P. Gupta, “Comparison of iris recognition algorithms,” in
36 *International Conference on Intelligent Sensing and Information Processing*, 2004.
]
- [1 P. Phillips, K. Bowyer and P. Flynn, “Comments on the CASIA version 1.0 Iris Data
37 Set,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 10,
] pp. 1869-1870, Oct. 2007.
- [1 Panasonic, “Learn about Panasonic's BM-ET100US,” Panasonic, 09 2001. [Online].
38 Available:
] [http://catalog2.panasonic.com/webapp/wcs/stores/servlet/ModelDetail?storeId=11201&
catalogId=13051&itemId=63725&catGroupId=16817&surfModel=BM-
ET100US&displayTab=O](http://catalog2.panasonic.com/webapp/wcs/stores/servlet/ModelDetail?storeId=11201&catalogId=13051&itemId=63725&catGroupId=16817&surfModel=BM-ET100US&displayTab=O). [Accessed 06 2011].
- [1 H. Proença, G. Santos, M. Bernardo and P. Fiadeiro, “Iris Recognition: Preliminary
39 Assessment about the Discriminating Capacity of Visible Wavelength Data,” in *IEEE*
] *International Symposium on Multimedia*, Taichung, 2010.
- [1 S. Li, L. Qian and Y. Xin, “Study on Algorithm of Eyelash Occlusions Detection Based
40 on Endpoint Identification,” in *3rd International Workshop on Intelligent Systems and*
] *Applications (ISA)*, Wuhan, 2011.
- [1 G. Xu, Y. Wang, M. Zhou and X. Huang, “A Reflection and Occlusion Robust Eye
41 Center Searching Algorithm,” in *Chinese Conference on Pattern Recognition (CCPR)*,
] Chongqing, 2010.
- [1 Y. Yan and M. Xie, “Eyelid and Eyelash Detection Method Based on Morphology,” in
42 *The 2nd International Conference on Computer and Automation Engineering (ICCAE)*,
] Singapore, 2010.

[1 Association BioSecure, "BioSecure," Association BioSecure, 2012. [Online].
43 Available: <http://biosecure.it-sudparis.eu/AB/>. [Accessed 08 2012].

]

[1 D. M. Monro, S. Rakshit and D. Zhang, "DCT-Based Iris Recognition," *IEEE*
44 *Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 4, pp. 586 -
] 595, 2007.

[1 M. Abhiram, C. Sadhu, K. Manikantan and S. Ramachandran, "Novel DCT based
45 feature extraction for enhanced Iris Recognition," in *International Conference on*
] *Communication, Information & Computing Technology (ICCICT)*, Mumbai, 2012.

[1 F. Arnia, K. Munadi, Roslidar, M. Fujiyoshi and H. Kiya, "Improved Iris matching
46 technique using reduced sized of ordinal measure of DCT coefficients," in *IEEE 17th*
] *International Symposium on Consumer Electronics (ISCE)*, Hsinchu , 2013.

[1 N. Ahmed, T. Nataraja and K. R. Rao, "Discrete Cosine Transform," *IEEE*
47 *Transactions on Computers*, vol. 23, no. January, pp. 90-93, 1974.

]

[1 P. kovesi, "MATLAB and Octave Functions for Computer Vision and Image
48 Processing," 2009. [Online]. Available:
] <http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/index.html>. [Accessed 26 11
2010].

[1 R. Chan, "New parallel Hough transform for circles," *IEE Proceedings-E Computers*
49 *and Digital Techniques*, vol. 138, no. 5, pp. 335 - 344, 09 1991.

]

[1 P. Chochia, "Automatic gray scale correction of video data," *SPIE Image Processing*
50 *and Computer Optics* , vol. 2363, pp. 83-88, 1994.

]

[1 I. Mesecan, E. U. T. T. A. Comput. Eng., A. Eleyan and B. Karlik, "Sift-based iris

51 recognition using sub-segments,” in *International Conference on Technological*
] *Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, Konya,
2013.

[1 J.-g. Ko, D. Electron. & Telecommun. Res. Inst., Y.-H. Gil and J.-H. Yoo, “Iris
52 Recognition using Cumulative SUM based Change Analysis,” in *International*
] *Symposium on Intelligent Signal Processing and Communications*, Yonago, 2006.

[1 Y. Du, B. Bonney, R. Ives, D. Etter and R. Schultz, “Analysis of partial iris recognition
53 using a 1D approach,” in *IEEE International Conference on Acoustics, Speech, and*
] *Signal Processing*, 2005.

[1 L. Ma, T. Tan, Y. Wang and D. Zhang, “Local Intensity Variation Analysis for Iris
54 Recognition,” *Pattern Recognition*, vol. 37, no. 6, pp. 1287-1298, 2004.

]

[1 K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi and H. Nakajima, “An Efficient Iris
55 Recognition Algorithm Using Phase-Based Image Matching,” in *Proceedings of the*
] *Internationall Conference on Image Processing*, 2005.

[1 C. L. Tisse, L. Martin, L. Torres and M. Robert, “Person Identification Technique
56 Using Human Iris Recognition,” *Proceedings Of Vision Interface*, pp. 294-299, 2002.

]

[1 M. Grundland and N. Dodgson, “AUTOMATIC CONTRAST ENHANCEMENT BY
57 HISTOGRAM WARPING,” *Computer Vision and Graphics Computational Imaging*
] *and Vision*, vol. 32, pp. 293-300, 2006.

[1 L. McMillan, “Comp 136 -- Circle-Drawing Algorithms,” 17 09 1996. [Online].
58 Available: <http://www.cs.unc.edu/~mcmillan/comp136/Lecture7/circle.html>. [Accessed
] 28 09 2010].

[1 J. Daugman, “John Daugman's webpage, Cambridge University, Computer Laboratory,
59 Cambridge UK,” University of Cambridge, 2009. [Online]. Available:

] <http://www.cl.cam.ac.uk/~jgd1000/>. [Accessed 02 08 2010].